NLTK DOCUMENTATION:

The Natural Language Toolkit (NLTK) is a comprehensive and user-friendly Python library for processing natural language (text). Developed initially as an educational tool, NLTK has grown into a robust library for building complex text analysis and natural language processing (NLP) applications. It is widely used for tasks involving human language data.

In [5]:
```
#import Libraries
import os
import nltk
```

In [6]:
```
AI = '''Artificial Intelligence refers to the intelligence of machines. Thi
humans and animals. With Artificial Intelligence, machines perform function
problem-solving. Most noteworthy, Artificial Intelligence is the simulation
It is probably the fastest-growing development in the World of technology a
AI could solve major challenges and crisis situations.'''
```

In [7]:
```
AI
```

Out[7]:
```
'Artificial Intelligence refers to the intelligence of machines. This is i
n contrast to the natural intelligence of \nhumans and animals. With Artif
icial 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. Fur
thermore, many experts believe\nAI could solve major challenges and crisis
situations.'
```

In [4]:
```
type(AI)
```

Out[4]:
```
str
```

Tokenization

Tokenization is the process of breaking down text into smaller components, such as words or sentences. Tokenization is often a first step in NLP, as it segments text into manageable parts that can be further processed.

1] Word Tokenization:

Splits sentences into individual words.

In [5]:
```
from nltk.tokenize import word_tokenize
```

In [6]:
```
AI_tokens = word_tokenize(AI) #Splits sentences into individual words.
AI_tokens
```

Out[6]:
```
['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',
 '.']
```

In [7]:
```
len(AI_tokens) #Length
```

Out[7]:
```
81
```

2] Sentence Tokenization:

Splits text into individual sentences.

In [3]:
```
from nltk.tokenize import sent_tokenize
```

In [8]:
```
AI_sent = sent_tokenize(AI)#Splits text into individual sentences.
AI_sent
```

Out[8]:
```
['Artificial Intelligence refers to the intelligence of machines.',
 'This is in contrast to the natural intelligence of \nhumans and animal
s.',
 'With Artificial Intelligence, machines perform functions such as learnin
g, planning, reasoning and \nproblem-solving.',
 'Most noteworthy, Artificial Intelligence is the simulation of human inte
lligence by machines.',
 'It is probably the fastest-growing development in the World of technolog
y and innovation.',
 'Furthermore, many experts believe\nAI could solve major challenges and c
risis situations.']
```

In [10]:
```
len(AI_sent)
```

Out[10]:
```
6
```

In [11]:
```
AI
```

Out[11]:
```
'Artificial Intelligence refers to the intelligence of machines. This is i
n contrast to the natural intelligence of \nhumans and animals. With Artif
icial 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. Fur
thermore, many experts believe\nAI could solve major challenges and crisis
situations.'
```

3] Blankline Tokenization:

Blankline tokenization is a technique used to split text based on blank lines.

In [12]:
```
# GiVE YOU HOW MANY PARAGRAPH

from nltk.tokenize import blankline_tokenize
AI_blank =  blankline_tokenize(AI)
```

In [13]:
```
len(AI_blank)
```

Out[13]:
```
1
```

To generate unigrams, bigrams, and trigrams using NLTK.

In [14]:
```
# NEXT WE WILL SEE HOW WE WILL USE UNI-GRAM,BI-GRAM,TRI-GRAM USING NLTK

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

In [15]:
```
string = 'the best and most beautifull thing in the world cannot be seen or
quotes_tokens = nltk.word_tokenize(string)
```

Unigram:

A unigram is simply each individual word in a text.

In [16]:
```
quotes_tokens
```

Out[16]:
```
['the',
 'best',
 'and',
 'most',
 'beautifull',
 'thing',
 'in',
 'the',
 'world',
 'can',
 'not',
 'be',
 'seen',
 'or',
 'even',
 'touched',
 ',',
 'they',
 'must',
 'be',
 'felt',
 'with',
 'heart']
```

In [17]:
```
len(quotes_tokens)
```

Out[17]:
```
23
```

Bigrams:

A bigram is a sequence of two adjacent words. NLTK's bigrams function can help us generate bigrams from a list of tokens.

In [18]:
```
quotes_bigrams = list(nltk.bigrams(quotes_tokens))
quotes_bigrams
```

Out[18]:
```
[('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')]
```

In [19]:
```
len(quotes_bigrams)
```

Out[19]:
```
22
```

Trigrams:

A trigram is a sequence of three adjacent words. NLTK’s trigrams function generates trigrams from a list of tokens.

In [20]:
```
quotes_trigrams = list(nltk.trigrams(quotes_tokens))
quotes_trigrams
```

Out[20]:
```
[('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')]
```

In [21]:
```
len(quotes_trigrams)
```

Out[21]:
```
21
```

Ngrams:

For generating n-grams of arbitrary length (n), NLTK also provides an ngrams function, which lets you specify the value of n.

In [22]:
```
#it has given n-gram of length 4

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

Out[22]:
```
[('the', 'best', 'and', 'most'),
 ('best', 'and', 'most', 'beautifull'),
 ('and', 'most', 'beautifull', 'thing'),
 ('most', 'beautifull', 'thing', 'in'),
 ('beautifull', 'thing', 'in', 'the'),
 ('thing', 'in', 'the', 'world'),
 ('in', 'the', 'world', 'can'),
 ('the', 'world', 'can', 'not'),
 ('world', 'can', 'not', 'be'),
 ('can', 'not', 'be', 'seen'),
 ('not', 'be', 'seen', 'or'),
 ('be', 'seen', 'or', 'even'),
 ('seen', 'or', 'even', 'touched'),
 ('or', 'even', 'touched', ','),
 ('even', 'touched', ',', 'they'),
 ('touched', ',', 'they', 'must'),
 (',', 'they', 'must', 'be'),
 ('they', 'must', 'be', 'felt'),
 ('must', 'be', 'felt', 'with'),
 ('be', 'felt', 'with', 'heart')]
```

In [23]:
```
len(quotes_ngrams)
```

Out[23]:
```
20
```

In [24]:
```
quotes_ngrams_1 = list(nltk.ngrams(quotes_tokens,5))
quotes_ngrams_1
```

Out[24]:
```
[('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')]
```

In [25]:
```
len(quotes_ngrams_1)
```

Out[25]:
```
19
```

In [26]:
```
quotes_ngrams = list(nltk.ngrams(quotes_tokens,9))
quotes_ngrams
```

Out[26]:
```
[('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')]
```

In [27]:
```
len(quotes_ngrams)
```

Out[27]:
```
15
```

Stemming and Lemmatization

Stemming and lemmatization are used to reduce words to their root forms, which helps in normalizing text and reducing redundancy.

1]PorterStemmer

The Porter Stemmer is one of the most popular stemming algorithms in natural language processing. It was developed by Martin Porter in 1980 and is designed to reduce words to their root or base form, known as the stem. This process involves stripping common suffixes (like -ing, -ed, -ly, -s) to yield the stem word, which helps in text normalization.

In [28]:
```
# Next we need to make some changes in tokens and that is called as stemmin
# also we will see some root form of the word & limitation of the word

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

In [29]:
```
pst.stem('having')  #stem will gives you the root form of the word
```

Out[29]:
```
'have'
```

In [30]:
```
pst.stem('affection')
```

Out[30]: 'affect'

In [31]:
```
pst.stem('affecting')
```

Out[31]: 'affect'

In [32]:
```
pst.stem('affect')
```

Out[32]: 'affect'

In [33]:
```
pst.stem('playing')
```

Out[33]: 'play'

In [34]:
```
word_to_stem = ['give','giving','given','gave']
for words in word_to_stem:
    print(words+ ':' + pst.stem(words))
```

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

In [35]:
```
pst.stem('playing')
```

Out[35]: 'play'

In [36]:
```
pst.stem('played')
```

Out[36]: 'play'

In [37]:
```
word_to_stem = ['give','giving','given','gave','playing','played','loving',
# i am giving these different words to stem, using porter stemmer we get th

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

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

```
give:give
giving:give
given:given
gave:gave
playing:play
played:play
loving:love
thinking:think
final:final
finally:final
finalized:final
finalizing:final
```

2] lencastemmer

The Lancaster Stemmer is another stemming algorithm available in NLTK, known for its aggressive stemming approach. Developed as an alternative to the Porter Stemmer, the Lancaster Stemmer is simpler and often produces shorter stems, but it's also more aggressive.

In [38]:
```
#another stemmer known as lencastemmer stemmer and lets see what the differ
#stem the same thing using lencastemmer

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

# lancasterstemmer is more aggresive then the porterstemmer
```

```
give:giv
giving:giv
given:giv
gave:gav
playing:play
played:play
loving:lov
thinking:think
final:fin
finally:fin
finalized:fin
finalizing:fin
```

In [39]:
```
word_to_stem = ['give','giving','given','gave','playing','played','loving',
# i am giving these different words to stem, using porter stemmer we get th

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

```
give:give
giving:give
given:given
gave:gave
playing:play
played:play
loving:love
thinking:think
final:final
finally:final
finalized:final
finalizing:final
```

3] SnowballStemmer

The Snowball Stemmer is an algorithm for stemming words in various languages, which means it reduces words to their base or root form.

In [40]:
```
#we have another stemmer called as snowball stemmer lets see about this sno

from nltk.stem import SnowballStemmer
sbst = SnowballStemmer('english')
for words in word_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 l
```

```
give:give
giving:give
given:given
gave:gave
playing:play
played:play
loving:love
thinking:think
final:final
finally:final
finalized:final
finalizing:final
```

In [41]:
```
#sometime stemming does not work & lets say e.g - fish,fishes & fishing all
#one hand stemming will cut the end & lemmatization will take into the morp

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 wordn
```

In [42]:
```
word_to_stem
```

Out[42]:
```
['give',
 'giving',
 'given',
 'gave',
 'playing',
 'played',
 'loving',
 'thinking',
 'final',
 'finally',
 'finalized',
 'finalizing']
```

In [43]:
```
#word_lem.lemmatize('corpora') #we get output as corpus

#refers to a collection of texts. Such collections may be formed of a singl


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

```
give:give
giving:giving
given:given
gave:gave
playing:playing
played:played
loving:loving
thinking:thinking
final:final
finally:finally
finalized:finalized
finalizing:finalizing
```

In [44]:
```
pst.stem('final')
```

Out[44]: 'final'

In [45]:
```
lst.stem('finally')
```

Out[45]: 'fin'

In [46]:
```
sbst.stem('finalized')
```

Out[46]: 'final'

In [47]:
```
lst.stem('finalized')
```

Out[47]: 'fin'

Stopwords

In [48]:
```
# there is other concept called POS (part of speech) which deals with subje
# STOPWORDS = i, is, as,at, on, about & nltk has their own list of stopewor

from nltk.corpus import stopwords
```

In [49]:
```
stopwords.words('english')
```

Out[49]:
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
```

In [50]:
```
len(stopwords.words('english'))
```

Out[50]:
```
179
```

In [51]:
```
stopwords.words('spanish')
```

Out[51]:
```
['de',
 'la',
 'que',
 'el',
 'en',
 'y',
 'a',
 'los',
 'del',
 'se',
 'las',
 'por',
 'un',
 'para',
 'con',
 'no',
 'una',
 'su',
 'al',
```

In [52]:
```
len(stopwords.words('spanish'))
```

Out[52]:
```
313
```

In [53]:
```
stopwords
```

Out[53]:
```
<WordListCorpusReader in 'C:\\Users\\rutik\\AppData\\Roaming\\nltk_data\\corpora\\stopwords'>
```

In [ ]:

In [55]:
```
# first we need to compile from re module to create string that matched any

import re
punctuation = re.compile(r'[-.?!,;:()|0-9]')
 #now i am going to create to empty list and append the word without any pu
```

In [56]:
```
punctuation
```

Out[56]:
```
re.compile(r'[-.?!,;:()|0-9]', re.UNICODE)
```

In [57]:
```
AI
```

Out[57]:
```
'Artificial Intelligence refers to the intelligence of machines. This is i
n contrast to the natural intelligence of \nhumans and animals. With Artif
icial 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. Fur
thermore, many experts believe\nAI could solve major challenges and crisis
situations.'
```

In [58]:
```
AI_tokens
```

Out[58]:
```
['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',
 '.']
```

In [59]:
```
len(AI_tokens)
```

Out[59]: 81

Part-of-Speech (POS) Tagging

#POS [part of sppech] is always talking about grammaticaly type of the word called verbs, noun, adjective, proverb,

#how the word will function in grammatically within the sentence, a word can have more then one pos based on context in which it will use

#so lets see some pos tags & description, so pos tags are usualy used to descrie weather te word is used for noun,adjective,pronoun, propernoun, singular, plural, is it symbol or is it adverb

#in this slide we have so many tags along with their description with different tags

#this tags are beginning from coordinating conjunction to whadverb & lets understand about one of the example

#next we will see how we will implement this POS in our text

In [60]:
```
# 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
```

Out[60]: ['kathy', 'is', 'a', 'natural', 'when', 'it', 'comes', 'to', 'drawing']

In [61]:
```
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')]
```

In [62]:
```
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')]
```

Named Entity Recognition (NER)

```
# 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
```

In [63]:
```
from nltk import ne_chunk
```

In [64]:
```
NE_sent = 'The US president stays in the WHITEHOUSE'
```

```
# IN NLTK also we have syntax- set of rules,principals & process
# lets understand set of rules & that will indicates the syntax tree & in
the real time also you have build this type of tree from the sentenses

# now lets understand the important concept called CHUNKING using the
sentence structure
# chunking means grouping of words into chunks & lets understand the
example of chunking
```

```
# chunking will help to easy process the data
```

In [65]:
```
NE_tokens = word_tokenize(NE_sent)

#after tokenize need to add the pos tags
NE_tokens
```

Out[65]:
```
['The', 'US', 'president', 'stays', 'in', 'the', 'WHITEHOUSE']
```

In [66]:
```
NE_tags = nltk.pos_tag(NE_tokens)
NE_tags
```

Out[66]:
```
[('The', 'DT'),
 ('US', 'NNP'),
 ('president', 'NN'),
 ('stays', 'NNS'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('WHITEHOUSE', 'NNP')]
```

In [67]:
```
#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))
```

In [68]:
```
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
```

Out[68]:
```
[('the', 'DT'),
 ('big', 'JJ'),
 ('cat', 'NN'),
 ('ate', 'VBD'),
 ('the', 'DT'),
 ('little', 'JJ'),
 ('mouse', 'NN'),
 ('who', 'WP'),
 ('was', 'VBD'),
 ('after', 'IN'),
 ('fresh', 'JJ'),
 ('cheese', 'NN')]
```

WordCloud

In [72]:
```
#Libraries
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

In [70]:
```
pip install Wordcloud
```

```
Defaulting to user installation because normal site-packages is not writea
ble
Collecting Wordcloud
  Downloading wordcloud-1.9.3-cp310-cp310-win_amd64.whl (299 kB)
     ------------------------------------ 300.0/300.0 kB 617.8 kB/s eta 0:
00:00
Requirement already satisfied: numpy>=1.6.1 in c:\programdata\anaconda3\li
b\site-packages (from Wordcloud) (1.23.5)
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib
\site-packages (from Wordcloud) (3.7.0)
Requirement already satisfied: pillow in c:\programdata\anaconda3\lib\site
-packages (from Wordcloud) (10.3.0)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\li
b\site-packages (from matplotlib->Wordcloud) (0.11.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\programdata\anaconda
3\lib\site-packages (from matplotlib->Wordcloud) (1.0.5)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anacond
a3\lib\site-packages (from matplotlib->Wordcloud) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3
\lib\site-packages (from matplotlib->Wordcloud) (22.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\programdata\anac
onda3\lib\site-packages (from matplotlib->Wordcloud) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anacond
a3\lib\site-packages (from matplotlib->Wordcloud) (4.25.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\programdata\anaconda
3\lib\site-packages (from matplotlib->Wordcloud) (3.0.9)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\si
te-packages (from python-dateutil>=2.7->matplotlib->Wordcloud) (1.16.0)
Installing collected packages: Wordcloud
Successfully installed Wordcloud-1.9.3
Note: you may need to restart the kernel to use updated packages.

  WARNING: The script wordcloud_cli.exe is installed in 'C:\Users\rutik\Ap
pData\Roaming\Python\Python310\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress thi
s warning, use --no-warn-script-location.
```

In [73]:
```
# Create a list of word

text = ("Python Python Python Matplotlib Matplotlib Seaborn Network Plot Vi
```

In [74]:
```
text
```

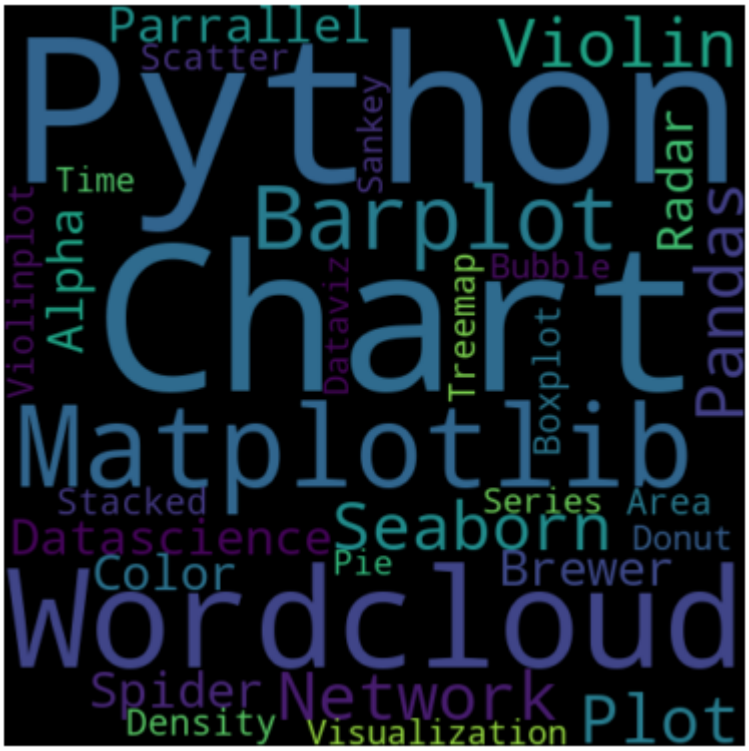
Out[74]:
```
'Python Python Python Matplotlib Matplotlib Seaborn Network Plot Violin Ch
art Pandas Datascience Wordcloud Spider Radar Parrallel Alpha Color Brewer
Density Scatter Barplot Barplot Boxplot Violinplot Treemap Stacked Area Ch
art Chart Visualization Dataviz Donut Pie Time-Series Wordcloud Wordcloud
Sankey Bubble'
```

In [75]:
```
text
```

Out[75]:
```
'Python Python Python Matplotlib Matplotlib Seaborn Network Plot Violin Ch
art Pandas Datascience Wordcloud Spider Radar Parrallel Alpha Color Brewer
Density Scatter Barplot Barplot Boxplot Violinplot Treemap Stacked Area Ch
art Chart Visualization Dataviz Donut Pie Time-Series Wordcloud Wordcloud
Sankey Bubble'
```

In [76]:
```
# Create the wordcloud object
wordcloud =WordCloud(width=480,height=480,margin=0).generate(text)
```

In [77]:
```
# Display the generated image:
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()
```



In [78]:
```
text1 = ('Rutika Gitanjali vrushali nikita kajal ppooja pranav durgesh kuna
```

In [79]:
```
text1
```

Out[79]:
```
'Rutika Gitanjali vrushali nikita kajal ppooja pranav durgesh kunal nilesh
sanket priti gayatri shubham noman danny  '
```

In [86]:
```
wordcloud = WordCloud(width=500,height=500,margin=0).generate(text1)
```

In [87]:
```
plt.imshow(wordcloud,interpolation = 'bilinear')
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()
```

In [ ]:

In [ ]:

In [ ]: