In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\MR.GODHADE\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[2]:
```
True
```

## NLTK

NLTK is a leading platform for building Python programs to work with human language data.

It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum

In [52]:
```
paragraph = 'this is the NLP NLP session is going on  and finally we ar
```

## nltk.sent_tokenize()

This converts the paragraph into sentences as we seen above example

In [53]:
```
#Tokenization
nltk.sent_tokenize(paragraph)
```

Out[53]:
```
['this is the NLP NLP session is going on  and finally we are happy.',
 'Is everyone happy?',
 'I think everyone is happy']
```

## nltk.word_tokenize()

This converts the paraghraph into words

In [54]:
```
1 nltk.word_tokenize(paragraph)
```

Out[54]:
```
['this',
 'is',
 'the',
 'NLP',
 'NLP',
 'session',
 'is',
 'going',
 'on',
 'and',
 'finally',
 'we',
 'are',
 'happy',
 '.',
 'Is',
 'everyone',
 'happy',
 '?',
 'I',
 'think',
 'everyone',
 'is',
 'happy']
```

## Stemming

Stemming is a text preprocessing technique used in natural language processing (NLP) to reduce words to their root or base form.

In [55]:
```
1 from IPython import display
2 display.Image('download.PNG')
```

Out[55]:


In [56]:
```
1 from nltk.stem import PorterStemmer
```

### PorterStemmer

It help for stemming

In [57]:
```
stemmer = PorterStemmer()
```

In [58]:
```
stemmer.stem('going')
```

Out[58]: 'go'

In [59]:
```
stemmer.stem('finally')
```

Out[59]: 'final'

## This is the problem with stemming

In [60]:
```
stemmer.stem('organization')
```

Out[60]: 'organ'

In [61]:
```
stemmer.stem('history')
```

Out[61]: 'histori'

In [62]:
```
stemmer.stem('happy')
```

Out[62]: 'happi'

## Above Problem is overcome by Lemmatization

### Lemmatization

is a text pre-processing technique used in natural language processing (NLP) models to break a word down to its root meaning to identify similarities.

In [63]:
```
from nltk.stem import WordNetLemmatizer
```

In [64]:
```
lemmatizer = WordNetLemmatizer()
```

In [65]:
```
lemmatizer.lemmatize('organizations',pos='n')
```

Out[65]: 'organization'

In [66]:
```
lemmatizer.lemmatize('eating',pos='v')
```

Out[66]: 'eat'

In [67]:
```
lemmatizer.lemmatize('sweets')
```

Out[67]: 'sweet'

In [68]:
```
lemmatizer.lemmatize('better',pos='a')
```

Out[68]:
```
'good'
```

## Parameter pos

```
 The Part Of Speech tag. Valid options are `"n"` for nouns,
                                           `"v"` for verbs,
                                           `"a"` for adjectives,
                                           `"r"` for adverbs and
                                           `"s"`for satellite
adjectives.
```

## Differrence Between Stemming & Lemmatization

In [70]:
```
paragraph = 'this is the NLP NLP session is going on  and finally we ar
```

In [71]:
```
sentencess = nltk.sent_tokenize(paragraph)
```

In [72]:
```
sentencess
```

Out[72]:
```
['this is the NLP NLP session is going on  and finally we are happy.',
 'Is everyone happy?',
 'I think everyone is happy']
```

## Applied stemming on paragraph

In [73]:
```
corpus = []
for i in range(len(sentencess)):
    words = nltk.word_tokenize(sentencess[i])
    words = [stemmer.stem(word) for word in words]
    corpus.append(' '.join(words))
```

In [74]:
```
corpus   # Meaning may chane
```

Out[74]:
```
['thi is the nlp nlp session is go on and final we are happi .',
 'is everyon happi ?',
 'i think everyon is happi']
```

## Applied Lemmatization on paragraph

In [75]:
```
corpus = []
for i in range(len(sentencess)):
    words = nltk.word_tokenize(sentencess[i])
    words = [lemmatizer.lemmatize(word) for word in words]
    corpus.append(' '.join(words))
```

In [76]:
```
corpus    # Meaningful words
```

Out[76]:
```
['this is the NLP NLP session is going on and finally we are happy .',
 'Is everyone happy ?',
 'I think everyone is happy']
```

## StopWords

In natural language processing (NLP), "stopwords" are words that are commonly used in a language but are often filtered out or ignored during text processing because they are considered to have little or no meaningful information.

Examples of stopwords in English include words like "the," "and," "in," "is," "at," "it," "on," and so on.

In [77]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\MR.GODHADE\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[77]:
```
True
```

In [78]:
```
from nltk.corpus import stopwords
```

In [79]:
```
stop_words = stopwords.words('english')
```

In [80]:
```
stop_words
```

Out[80]:
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
```

## Remove stopwords from paragraph

In [81]:
```
paragraph = 'this is the NLP session is going on  and finally we are ha
```

In [82]:
```
import re
```

In [83]:
```
corpus = []
for i in range(len(sentencess)):
    text = re.sub('[^a-zA-Z0-9]'," ",sentencess[i])
    text = text.lower()
    words = text.split()
    words = [lemmatizer.lemmatize(word) for word in words if not word i
    print(words)
    corpus.append(' '.join(words))
```

```
['nlp', 'nlp', 'session', 'going', 'finally', 'happy']
['everyone', 'happy']
['think', 'everyone', 'happy']
```

In [ ]:
```

```

In [84]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

In [85]:
```
cv = CountVectorizer()
```

In [86]:
```
# Applied BOW()
cv.fit_transform(corpus).toarray()
```

Out[86]:
```
array([[0, 1, 1, 1, 2, 1, 0],
       [1, 0, 0, 1, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 1]], dtype=int64)
```

## binary = True

In [88]:
```
cv = CountVectorizer(binary = True)
```

In [89]:
```
cv.fit_transform(corpus).toarray()  # 2 is converted 1
```

Out[89]:
```
array([[0, 1, 1, 1, 1, 1, 0],
       [1, 0, 0, 1, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 1]], dtype=int64)
```

In [87]:
```
cv.vocabulary_
```

Out[87]:
```
{'nlp': 4,
 'session': 5,
 'going': 2,
 'finally': 1,
 'happy': 3,
 'everyone': 0,
 'think': 6}
```

## N-gram

In [97]:
```
cv = CountVectorizer(ngram_range=(2,2))
```

In [98]:
```
cv.fit_transform(corpus)
```

Out[98]:
```
<3x7 sparse matrix of type '<class 'numpy.int64'>'
        with 8 stored elements in Compressed Sparse Row format>
```

In [99]:
```
cv.vocabulary_
```

Out[99]:
```
{'nlp nlp': 3,
 'nlp session': 4,
 'session going': 5,
 'going finally': 2,
 'finally happy': 1,
 'everyone happy': 0,
 'think everyone': 6}
```

## Term Frequency -Inverse Document Frequency(TF-IDF)

In [100]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [101]:
```
tf_idf = TfidfVectorizer()
```

In [103]:
```
tf_idf.fit_transform(corpus).toarray()
```

Out[103]:
```
array([[0.        , 0.36888498, 0.36888498, 0.21786941, 0.73776997,
        0.36888498, 0.        ],
       [0.78980693, 0.        , 0.        , 0.61335554, 0.        ,
        0.        , 0.        ],
       [0.54783215, 0.        , 0.        , 0.42544054, 0.        ,
        0.        , 0.72033345]])
```

In [104]:
```
tf_idf.vocabulary_
```

Out[104]:
```
{'nlp': 4,
 'session': 5,
 'going': 2,
 'finally': 1,
 'happy': 3,
 'everyone': 0,
 'think': 6}
```

### new test data 1 ¶

In [109]:
```
data = ['i want to have food']   # no single word is related

cv.transform(data).toarray()
```

Out[109]:
```
array([[0, 0, 0, 0, 0, 0, 0]], dtype=int64)
```

### new test data 2

In [110]:
```
data = ['i want to attend NLP session']  # NLP word is related to Trair

cv.transform(data).toarray()
```

Out[110]:
```
array([[0, 0, 0, 0, 1, 0, 0]], dtype=int64)
```

In [ ]:
```

```