**NLTK library:**

Stand for Natural language toolkit

Operation that we can performed

1. **Tokenization**

Conversion of the paragraph into individual sentences or into words. It can be performed in two ways:

* Sentence tokenizes
* Word tokenizes

**Syntax:** From nltk.tokenize import sent\_tokenize, and word\_tokenize

1. **Filtering stop words**

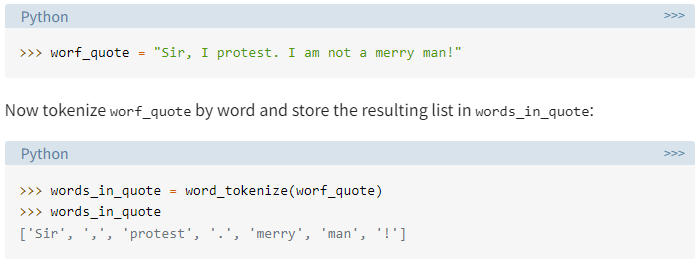
Stop words are the words that you want to ignore, so you filter them out of your text when u r processing it. Like as in, as, an, are often called as stop words. Basically, these words don’t convey any meaning in a sentence these words are just adding to amplify the meaning or the action of the subject.

**Syntax:** nltk.downlaod(“Stopwards”)

from nltk.corpus import stopwards

from nltk.tokenize import word\_tokenize

E.g.



1. **Stemming**

In stemming a word is reduces to its root word such as helping and helper having a root word help so in stemming u get an output as help. NLTK has many stemmers, but the most common is porter stemmer.

**Syntax:**

From nltk.stem import porterstemmer

From nltk.tokenize import word\_tokenize // before stemming u need to perform tokenization in order to reduce the sentence or paragraph into individual words.

1. **Parts of speech also known as POS tagging**

Identifying the POS of each word in a sentence is done using POS tagging. It helps you to classify each word as noun, pronoun, adjective, adverb and many more.

**Syntax:**

Import nltk

From nltk. tokenize import word\_tokenize

Nltk.POS\_tag(variable storing the string)

1. **Lemmatizing**

It is same as stemming but in stemming there are some drawbacks that can be solved using lemmatizing for e.g. when you apply stemming on word such as discovery u got the root word as discoveri and on discovering u got discover in both cases u got different root words. This is known as understemming. Here, lemmatizing come into existence in this, the word is reducing to its core meaning but it gives a complete english word that makes sense.

**Syntax:**

From nltk.stem import wordNet Lemmatizer

1. **Chunking**

It is much similar like tokenization, while tokenization allows you to identify words and sentences, chunking allow you to identify phrases.

Note: Here a phrase is a word or group of words that works as a single unit to perform a grammatical function. Noun phrases are built around a noun.

For apply chunking we first need to apply tokenization then POS tagging and then we can apply chunking.

Syntax:

From nltk.tokenize import word\_tokenize

From nltk import nltk.POS\_tagging

From nltk import nltk.RegexParser (variable storing the string)

**Chinking**

Chinking is used with chunking, while we include the pattern of the phrase in chunking. In chinking, we exclude the phrases hence in this a particular regular expression in grammar is created to remove a specific phrase or word.

For e.g.

“It's a dangerous business, Frodo, going out your door” in this particular string if u want to remove “dangerous” which is an adjective u create a regular expression to remove adjective as :

**Syntax**

grammar = """

Chunk: {<.\*>+}

}<JJ>{"""

The first rule of your grammar is {<.\*>+}. This rule has curly braces that face inward ({}) because it’s used to determine what patterns you want to include in you chunks. In this case, you want to include everything: <.\*>+.

The second rule of your grammar is }<JJ>{. This rule has curly braces that face outward (}{) because it’s used to determine what patterns you want to exclude in your chunks. In this case, you want to exclude adjectives: <JJ>.

**Named entity recognition**

Named entities are basically refers to the identification of a person, location, organisation, date, time, and some more fixed entities.

Nltk.ne\_chunk() function is used to recognizes the named entities.

**Getting text to analyze**

Before this first define a corpus, basically a corpus is a collection of text or sentences and NLTK provide several corpora covering almost everthing