NLP/NLTK Basics:

The text must be parsed to remove words, called **Tokenization**. Then the words need to be encoded as integers or floating-point values for use as input to a machine learning algorithm, called feature extraction (or **Vectorization**).

Bag-of-Words Model:

A simple and effective model for thinking about text documents in machine learning is called the Bag-of-Words Model, or BoW. The model is simple in that it throws away all of the order information in the words and focuses on the occurrence of words in a document. This can be done by assigning each word a unique number. Then any document we see can be encoded as a fixed-length vector with the length of the vocabulary of known words. The value in each position in the vector could be filled with a count or frequency of each word in the encoded document. This is the bag of words model, where we are only concerned with encoding schemes that represent what words are present or the degree to which they are present in encoded documents without any information about order. There are many ways to extend this simple method, both by better clarifying what a "*word*" is and in defining what to encode about each word in the vector.

N-grams (sets of consecutive words):

For example,

```
txt = ["He is ::having a great Time, at the park time?",
    "She, unlike most women, is a big player on the park's grass.",
    "she can't be going"]
```

With ngram_range = (1,2) it will take below features:

Every feature:
['big', 'going', 'grass', 'great', 'having', 'park', 'player', 'time', 'unlike', 'women']

For ngram_range = (1,2) means the algorithm will consider 2 words together while doing vectorization

['big', 'big player', 'going', 'grass', 'great', 'great time', 'having', 'having great', 'park', 'park grass', 'park time', 'player', 'player park', 'time', 'time park', 'unlike', 'unlike women', 'women', 'women big']

The scikit-learn library provides 3 different schemes to use for BoW model (https://www.kaggle.com/adamschroeder/countvectorizer-tfidfvectorizer-predict-comments)

CountVectorizer: CountVectorizer can lowercase letters, disregard punctuation and stopwords, but it can't LEMMATIZE or STEM

TfidfVectorizer: The goal of using term frequency – inverse term frequency is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus. It learns vocabulary and idf, return term-document

matrix. Convert a collection of raw documents to a matrix of TF-IDF features. Equivalent to CountVectorizer followed by TfidfTransformer. The token which appears maximum times, but it is also in all documents, has its idf the lowest. The tokens can have the most idf weight because they are the only tokens that appear in one document only. the more times a token appears in a document, the more weight it will have. However, the more documents the token appears in, it is 'penalized' and the weight is diminished.

Stemming: The idea of stemming is a sort of normalizing method. Many variations of words carry the same meaning, other than when tense is involved.

First, we're going to grab and define our stemmer:

```
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize

ps = PorterStemmer()
```

Now, let's choose some words with a similar stem, like:

```
example_words = ["python","pythoner","pythoning","pythoned","pythonly"]
```

Next, we can easily stem by doing something like:

```
for w in example_words:
    print(ps.stem(w))
```

Our output:

```
python
python
python
python
pythonli
```

Parts of Speech Tagging: This means labeling words in a sentence as nouns, adjectives, verbs...etc. Even more impressive, it also labels by tense, and more. (https://pythonprogramming.net/part-of-speech-tagging-nltk-tutorial/?completed=/stemming-nltk-tutorial/)

POS tag list:

CC	coordinating conjunction
CD	cardinal digit
DT	determiner
EX	existential there (like: "there is" ... think of it like "there exists")
FW	foreign word
IN	preposition/subordinating conjunction
JJ	adjective 'big'
JJR	adjective, comparative 'bigger'
JJS	adjective, superlative 'biggest'
LS	list marker 1)
MD	modal could, will
NN	noun, singular 'desk'
NNS	noun plural 'desks'
NNP	proper noun, singular 'Harrison'

NNPS	proper noun, plural	'Americans'
PDT	predeterminer	'all the kids'
POS	possessive ending	parent\'s
PRP	personal pronoun	I, he, she
PRP$	possessive pronoun	my, his, hers
RB	adverb	very, silently,
RBR	adverb, comparative	better
RBS	adverb, superlative	best
RP	particle	give up
TO	to	go 'to' the store.
UH	interjection	errrrrrrrm
VB	verb, base form	take
VBD	verb, past tense	took
VBG	verb, gerund/present participle	taking
VBN	verb, past participle	taken
VBP	verb, sing. present, non-3d	take
VBZ	verb, 3rd person sing. present	takes
WDT	wh-determiner	which
WP	wh-pronoun	who, what
WP$	possessive wh-pronoun	whose
WRB	wh-abverb	where, when

```
def process_content():
    try:
        for i in tokenized[:5]:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)
            print(tagged)

    except Exception as e:
        print(str(e))


process_content()
```

The output should be a list of tuples, where the first element in the tuple is the word, and the second is the part of speech tag. It should look like:

[('PRESIDENT', 'NNP'), ('GEORGE', 'NNP'), ('W.', 'NNP'), ('BUSH', 'NNP'), ('"S", 'POS'), ('ADDRESS', 'NNP'), ('BEFORE', 'NNP'), ('A', 'NNP'), ('JOINT', 'NNP'), ('SESSION', 'NNP'), ('OF', 'NNP'), ('THE', 'NNP'), ('CONGRESS', 'NNP'), ('ON', 'NNP'), ('THE', 'NNP'), ('STATE', 'NNP'), ('OF', 'NNP'), ('THE', 'NNP'), ('UNION', 'NNP'), ('January', 'NNP'), ('31', 'CD'), (',', ','), ('2006', 'CD'), ('THE', 'DT'), ('PRESIDENT', 'NNP'), (':', ':'), ('Thank', 'NNP'), ('you', 'PRP'), ('all', 'DT'), ('.', '.')] [('Mr.', 'NNP'), ('Speaker', 'NNP'), (',', ','), ('Vice', 'NNP'), ('President', 'NNP'), ('Cheney', 'NNP'), (',', ','), ('members', 'NNS'), ('of', 'IN'), ('Congress', 'NNP'),

Lemmatizing: A very similar operation to stemming is called lemmatizing. The major difference between these is, as you saw earlier, stemming can often create non-existent words, whereas lemmas are actual words. So, your root stem, meaning the word you end up with, is not something you can just look up in a dictionary, but you can look up a lemma.

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print(lemmatizer.lemmatize("cats"))
print(lemmatizer.lemmatize("cacti"))
print(lemmatizer.lemmatize("geese"))
print(lemmatizer.lemmatize("rocks"))
print(lemmatizer.lemmatize("python"))
print(lemmatizer.lemmatize("better", pos="a"))
print(lemmatizer.lemmatize("best", pos="a"))
print(lemmatizer.lemmatize("run"))
print(lemmatizer.lemmatize("run",'v'))
```

Here, we've got a bunch of examples of the lemma for the words that we use. The only major thing to note is that lemmatize takes a part of speech parameter, "pos." If not supplied, the default is "noun." This means that an attempt will be made to find the closest noun, which can create trouble for you. Keep this in mind if you use lemmatizing!