Question 2

If you use Jupyter notebook with NLTK and libraries installed plus the nltk book download, you are good to go. If you use Colab, insert a code chunk at the top of your notebook to install these items:

```
# import all the libraries associated with nltk
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('omw-1.4')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')
nltk.download('gutenberg')
nltk.download('genesis')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package inaugural to /root/nltk_data...
[nltk_data]   Package inaugural is already up-to-date!
[nltk_data] Downloading package nps_chat to /root/nltk_data...
[nltk_data]   Package nps_chat is already up-to-date!
[nltk_data] Downloading package webtext to /root/nltk_data...
[nltk_data]   Package webtext is already up-to-date!
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Package treebank is already up-to-date!
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
[nltk_data] Downloading package genesis to /root/nltk_data...
[nltk_data]   Package genesis is already up-to-date!
True
```

Question 3

Code cell: Each of the built-in 9 texts is an NLTK Text object. Look at the code for the Text object at this link: https://www.nltk.org/_modules/nltk/text.html. Look at the tokens() method. Extract the first 20 tokens from text1. List two things you learned about the tokens() method or Text objects in the text cell above this code cell.

```
from nltk.book import *
```

```
# get the first 20 tokens
text1.tokens[:20]
```

```
['[',
 'Moby',
 'Dick',
 'by',
 'Herman',
 'Melville',
 '1851',
 ']',
 'ETYMOLOGY',
 '.',
 '(',
 'Supplied',
 'by',
 'a',
 'Late',
 'Consumptive',
 'Usher',
 'to',
 'a',
 'Grammar']
```

Question 4

Look at the concordance() method in the API. Using the documentation to guide you, in code, print a concordance for text1 word 'sea', selecting only 5 lines.

```
#Get the concordance for text1
text1.concordance("sea",80,5)
```

```
Displaying 5 of 455 matches:
 shall slay the dragon that is in the sea ." -- ISAIAH " And what thing soever
 S PLUTARCH ' S MORALS . " The Indian Sea breedeth the most and the biggest fis
cely had we proceeded two days on the sea , when about sunrise a great many Wha
many Whales and other monsters of the sea , appeared . Among the former , one w
 waves on all sides , and beating the sea before him into a foam ." -- TOOKE '
```

Question 5

The count() method in accordance with the API counts the number of times a particular word appears within the text. Python's count method is a string that acts in a very same way, except

Python counts the number of times a particular text appears in the string, depending on what you pass as a parameter (i.e. characters, substrings, etc.). In short, Python's count method is an extension of the count() method in the NLTK API.

Question 6

Code cell: Using raw text of at least 5 sentences of your choice from any source (cite the source), save the text into a variable called raw_text. Using NLTK's word tokenizer, tokenize the text into variable 'tokens'. Print the first 10 tokens.

```
#print out the first 10 tokens of raw_text after storing the long string.
from nltk import word_tokenize


raw_text='Voldemort himself created his worst enemy, just as tyrants everywhere do! Have you
'Voldemort himself created his worst enemy, just as tyrants everywhere do! Have you any idea
```

```
    'Voldemort himself created his worst enemy, just as tyrants everywhere do! Have y
    ou any idea how much tyrants fear the people they oppress? All of them realize th
    at, one day, amongst their many victims, there is sure to be one who rises agains
```

```
tokens=word_tokenize(raw_text)
tokens[:10]
```

```
    ['Voldemort',
     'himself',
     'created',
     'his',
     'worst',
     'enemy',
     ',',
     'just',
     'as',
     'tyrants']
```

Question 7

Code cell: Using the same raw text, and NLTK's sentence tokenizer sent_tokenize(), perform sentence segmentation and display the sentences.

```
# separate the string raw_text into sentences
from nltk import sent_tokenize
```

```
sent=sent_tokenize(raw_text)
sent
```

```
['Voldemort himself created his worst enemy, just as tyrants everywhere do!',
 'Have you any idea how much tyrants fear the people they oppress?',
 'All of them realize that, one day, amongst their many victims, there is sure to be
one who rises against them and strikes back!']
```

Question 8

Code cell: Using NLTK's PorterStemmer(), write a list comprehension to stem the text. Display the list.

```
tokens=word_tokenize(raw_text)
porter=nltk.PorterStemmer()
[porter.stem(t) for t in tokens] #list comprehension to stem the text
```

```
['voldemort',
 'himself',
 'creat',
 'hi',
 'worst',
 'enemi',
 ',',
 'just',
 'as',
 'tyrant',
 'everywher',
 'do',
 '!',
 'have',
 'you',
 'ani',
 'idea',
 'how',
 'much',
 'tyrant',
 'fear',
 'the',
 'peopl',
 'they',
 'oppress',
 '?',
 'all',
 'of',
 'them',
 'realiz',
 'that',
 ',',
 'one',
 'day',
 ',',
 'amongst',
 'their',
 'mani',
 'victim',
 ',',
 'there',
 'is',
 'sure',
 'to',
 'be',
 'one',
 'who',
 'rise',
 'against',
 'them',
 'and',
 'strike',
 'back',
 '!']
```

Question 9

9. Code cell: Using NLTK's WordNetLemmatizer, write a list comprehension to lemmatize the text. Display the list. In the text cell above this code cell, list at least 5 differences you see in the stems verses the lemmas. You can just write them each on a line, like this: stem-lemma

:**Stemmed Words - Lemmatized Words**

voldemort-Voldemort

creat-created

hi-his

enemi-enemy

as-a

everywher-everywhere

have-Have

ani-any

peopl-people

all-All

realiz-realize

mani-many

```
wnl=nltk.WordNetLemmatizer()
[wnl.lemmatize(t) for t in tokens] #list comprehension to lemmatize or produce the lemma of t
```

```
['Voldemort',
 'himself',
 'created',
 'his',
 'worst',
 'enemy',
 ',',
 'just',
 'a',
 'tyrant',
 'everywhere',
 'do',
 '!',
 'Have',
 'you',
 'any',
 'idea',
 'how',
 'much',
 'tyrant',
 'fear',
 'the',
 'people',
 'they',
 'oppress',
 '?',
 'All',
 'of',
 'them',
 'realize',
 'that',
 ',',
 'one',
 'day',
 ',',
 'amongst',
 'their',
 'many',
 'victim',
 ',',
 'there',
 'is',
 'sure',
 'to',
 'be',
 'one',
 'who',
 'rise',
 'against',
 'them',
 'and',
 'strike',
 'back',
 '!']
```

Question 10 a-c The NLTK library's functionality works well in regards to the concordance, the token member list indicated by [:] as well as the Stemmer and Lemmatize methods. The token member list will print out the number of tokens you specify after the colon. If we wanted to get the concordance of the first 5 lines, all we have to do is specify the object (i.e. text1), the .concordance("sea", 80,5) where "sea is the string, 80 is the number of characters per line, and we output the first 5 lines. The Stemmer method is more aggressive in trimming down words as opposed to the Lemmatize method. These functionality features can be attributed to the well-organized NLTK Book and NLTK Documentation, both of which provide excellent tutorials to navigate through these features. In fact, some of the ways in which I can use NLTK library in the future involve: (1) Utilizing the concordance method to search for a specific string inside the text for a specific number of lines, if not the whole text, so that I can pinpoint the text that I may need to generate a Natural Language Processing model; (2) Utilizing the Stemmer and Lemmatize method to find similarities and differences among the prefixes, affixes, and suffixes of words; (3) Utilizing the tokens member list to output a list of tokens within the text to find words that may be complicated, confusing, etc.

+ Code + Text

✓ 0s completed at 11:27 PM ● ✕