---

*You are currently looking at **version 1.0** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the [Jupyter Notebook FAQ (https://www.coursera.org/learn/python-text-mining/resources/d9pwm)](https://www.coursera.org/learn/python-text-mining/resources/d9pwm) course resource.*

Assignment 2 - Introduction to NLTK

In part 1 of this assignment you will use nltk to explore the Herman Melville novel Moby Dick. Then in part 2 you will create a spelling recommender function that uses nltk to find words similar to the misspelling.

Part 1 - Analyzing Moby Dick

In [1]:
```
import nltk
nltk.download("book")
from nltk.book import *

# text1: Moby Dick by Herman Melville 1851
```

```
[nltk_data] Downloading collection 'book'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /home/jovyan/nltk_data...
[nltk_data]    |   Package abc is already up-to-date!
[nltk_data]    | Downloading package brown to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package brown is already up-to-date!
[nltk_data]    | Downloading package chat80 to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package chat80 is already up-to-date!
[nltk_data]    | Downloading package cmudict to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package cmudict is already up-to-date!
[nltk_data]    | Downloading package conll2000 to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package conll2000 is already up-to-date!
[nltk_data]    | Downloading package conll2002 to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package conll2002 is already up-to-date!
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package dependency_treebank is already up-to-date!
[nltk_data]    | Downloading package genesis to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package genesis is already up-to-date!
[nltk_data]    | Downloading package gutenberg to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package gutenberg is already up-to-date!
[nltk_data]    | Downloading package ieer to /home/jovyan/nltk_data...
[nltk_data]    |   Package ieer is already up-to-date!
[nltk_data]    | Downloading package inaugural to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package inaugural is already up-to-date!
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package movie_reviews is already up-to-date!
[nltk_data]    | Downloading package nps_chat to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package nps_chat is already up-to-date!
[nltk_data]    | Downloading package names to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package names is already up-to-date!
[nltk_data]    | Downloading package ppattach to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package ppattach is already up-to-date!
[nltk_data]    | Downloading package reuters to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package reuters is already up-to-date!
[nltk_data]    | Downloading package senseval to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package senseval is already up-to-date!
[nltk_data]    | Downloading package state_union to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package state_union is already up-to-date!
[nltk_data]    | Downloading package stopwords to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package swadesh to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package swadesh is already up-to-date!
[nltk_data]    | Downloading package timit to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package timit is already up-to-date!
[nltk_data]    | Downloading package treebank to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package treebank is already up-to-date!
[nltk_data]    | Downloading package toolbox to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package toolbox is already up-to-date!
[nltk_data]    | Downloading package udhr to /home/jovyan/nltk_data...
[nltk_data]    |   Package udhr is already up-to-date!
[nltk_data]    | Downloading package udhr2 to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package udhr2 is already up-to-date!
[nltk_data]    | Downloading package unicode_samples to
[nltk_data]    |     /home/jovyan/nltk_data...
```

```
[nltk_data]    |   Package unicode_samples is already up-to-date!
[nltk_data]    | Downloading package webtext to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package webtext is already up-to-date!
[nltk_data]    | Downloading package wordnet to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package wordnet is already up-to-date!
[nltk_data]    | Downloading package wordnet_ic to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package wordnet_ic is already up-to-date!
[nltk_data]    | Downloading package words to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package words is already up-to-date!
[nltk_data]    | Downloading package maxent_treebank_pos_tagger to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package maxent_treebank_pos_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package maxent_ne_chunker is already up-to-date!
[nltk_data]    | Downloading package universal_tagset to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package universal_tagset is already up-to-date!
[nltk_data]    | Downloading package punkt to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package punkt is already up-to-date!
[nltk_data]    | Downloading package book_grammars to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package book_grammars is already up-to-date!
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package city_database is already up-to-date!
[nltk_data]    | Downloading package tagsets to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package tagsets is already up-to-date!
[nltk_data]    | Downloading package panlex_swadesh to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package panlex_swadesh is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /home/jovyan/nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    |
[nltk_data]  Done downloading collection book
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

In [2]:

```
import nltk, collections
import pandas as pd
import numpy as np

# If you would like to work with the raw text you can use 'moby_raw'
with open('moby.txt', 'r') as f:
    moby_raw = f.read()

# If you would like to work with the novel in nltk.Text format you can use 'text1'
moby_tokens = nltk.word_tokenize(moby_raw)
text1 = nltk.Text(moby_tokens)
```

Example 1

How many tokens (words and punctuation symbols) are in text1?

This function should return an integer.

```
In [3]: def example_one():

            return len(nltk.word_tokenize(moby_raw)) # or alternatively len(text1)

        example_one()
Out[3]: 254989
```

Example 2

How many unique tokens (unique words and punctuation) does text1 have?

This function should return an integer.

```
In [4]: def example_two():

            return len(set(nltk.word_tokenize(moby_raw))) # or alternatively len(set(text1))

        example_two()
Out[4]: 20755
```

Example 3

After lemmatizing the verbs, how many unique tokens does text1 have?

This function should return an integer.

```
In [5]: from nltk.stem import WordNetLemmatizer

        def example_three():

            lemmatizer = WordNetLemmatizer()
            lemmatized = [lemmatizer.lemmatize(w,'v') for w in text1]

            return len(set(lemmatized))

        example_three()
Out[5]: 16900
```

Question 1

What is the lexical diversity of the given text input? (i.e. ratio of unique tokens to the total number of tokens)

This function should return a float.

```
In [6]: def answer_one():

            result = example_two() / example_one()

            return result

        answer_one()
Out[6]: 0.08139566804842562
```

Question 2

What percentage of tokens is 'whale'or 'Whale'?

This function should return a float.

```
In [7]: def answer_two():

            dist = nltk.FreqDist(text1)

            result = (dist['whale'] + dist['Whale']) / example_one()*100

            return result

        answer_two()
```

```
Out[7]: 0.4125668166077752
```

Question 3

What are the 20 most frequently occurring (unique) tokens in the text? What is their frequency?

This function should return a list of 20 tuples where each tuple is of the form `(token, frequency)`. The list should be sorted in descending order of frequency.

```
In [8]: def answer_three():

            result = nltk.FreqDist(text1).most_common(20)

            return result

        answer_three()
```

```
Out[8]: [(',', 19204),
         ('the', 13715),
         ('.', 7308),
         ('of', 6513),
         ('and', 6010),
         ('a', 4545),
         ('to', 4515),
         (';', 4173),
         ('in', 3908),
         ('that', 2978),
         ('his', 2459),
         ('it', 2196),
         ('I', 2097),
         ('!', 1767),
         ('is', 1722),
         ('--', 1713),
         ('with', 1659),
         ('he', 1658),
         ('was', 1639),
         ('as', 1620)]
```

Question 4

What tokens have a length of greater than 5 and frequency of more than 150?

This function should return an alphabetically sorted list of the tokens that match the above constraints. To sort your list, use `sorted()`

In [9]:
```
def answer_four():

    dist = nltk.FreqDist(text1)

    vocabulary = dist.keys()

    words = [w for w in vocabulary if len(w) > 5 and dist[w] > 150]

    result = sorted(words)

    return result

answer_four()
```

Out[9]:
```
['Captain',
 'Pequod',
 'Queequeg',
 'Starbuck',
 'almost',
 'before',
 'himself',
 'little',
 'seemed',
 'should',
 'though',
 'through',
 'whales',
 'without']
```

Question 5

Find the longest word in text1 and that word's length.

This function should return a tuple `(longest_word, length)`.

In [10]:
```
def answer_five():

    word_length = max(len(w) for w in text1)

    word = [w for w in text1 if len(w) == word_length]

    result = word[0], word_length

    return result

answer_five()
```

Out[10]:
```
("twelve-o'clock-at-night", 23)
```

Question 6

What unique words have a frequency of more than 2000? What is their frequency?

"Hint: you may want to use `isalpha()` to check if the token is a word and not punctuation."

This function should return a list of tuples of the form `(frequency, word)` sorted in descending order of frequency.

```
In [11]: def answer_six():

             dist = nltk.FreqDist(text1)

             words = [(dist[w],w) for w in set(text1) if w.isalpha() and dist[w] > 2000]

             result = sorted(words, reverse=True)

             return result

         answer_six()
```

```
Out[11]: [(13715, 'the'),
          (6513, 'of'),
          (6010, 'and'),
          (4545, 'a'),
          (4515, 'to'),
          (3908, 'in'),
          (2978, 'that'),
          (2459, 'his'),
          (2196, 'it'),
          (2097, 'I')]
```

Question 7

What is the average number of tokens per sentence?

This function should return a float.

```
In [12]: def answer_seven():

             result = np.mean([len(nltk.word_tokenize(sent)) for sent in nltk.sent_tokenize(moby_raw)])

             return result

         answer_seven()
```

```
Out[12]: 25.881952902963864
```

Question 8

What are the 5 most frequent parts of speech in this text? What is their frequency?

This function should return a list of tuples of the form `(part_of_speech, frequency)` sorted in descending order of frequency.

```
In [13]: def answer_eight():

             tags = nltk.pos_tag(text1)

             counts = collections.Counter((x[1] for x in tags))

             result = counts.most_common(5)

             return result

         answer_eight()
```

```
Out[13]: [('NN', 32730), ('IN', 28657), ('DT', 25867), (',', 19204), ('JJ', 17620)]
```

Part 2 - Spelling Recommender

For this part of the assignment you will create three different spelling recommenders, that each take a list of misspelled words and recommends a correctly spelled word for every word in the list.

For every misspelled word, the recommender should find find the word in `correct_spellings` that has the shortest distance*, and starts with the same letter as the misspelled word, and return that word as a recommendation.

*Each of the three different recommenders will use a different distance measure (outlined below).

Each of the recommenders should provide recommendations for the three default words provided: `['cormulent', 'incendenece', 'validrate']`.

In [14]:
```
from nltk.corpus import words

correct_spellings = words.words()
```

Question 9

For this recommender, your function should provide recommendations for the three default words provided above using the following distance metric:

Jaccard distance (https://en.wikipedia.org/wiki/Jaccard_index) on the trigrams of the two words.

This function should return a list of length three: `['cormulent_reccomendation', 'incendenece_reccomendation', 'validrate_reccomendation']`.

In [15]:
```
def answer_nine(entries=['cormulent', 'incendenece', 'validrate']):

    result = []

    for entry in entries:

        spelling = [x for x in correct_spellings if x[0] == entry[0] and len(x) >= 5]

        Jaccard_distance = [nltk.jaccard_distance(set(nltk.ngrams(entry,n=3)), set(nltk.ngrams(x,n=3))) for x
in spelling]

        result.append(spelling[np.argmin(Jaccard_distance)])

    return result


answer_nine()
```

Out[15]:
```
['corpulent', 'indecence', 'validate']
```

Question 10

For this recommender, your function should provide recommendations for the three default words provided above using the following distance metric:

Jaccard distance (https://en.wikipedia.org/wiki/Jaccard_index) on the 4-grams of the two words.

This function should return a list of length three: `['cormulent_reccomendation', 'incendenece_reccomendation', 'validrate_reccomendation']`.

```
In [16]: def answer_ten(entries=['cormulent', 'incendenece', 'validrate']):

             result = []

             for entry in entries:

                 spelling = [x for x in correct_spellings if x[0] == entry[0] and len(x) >= 5]

                 Jaccard_distance = [nltk.jaccard_distance(set(nltk.ngrams(entry,n=4)), set(nltk.ngrams(x,n=4))) for x
         in spelling]

                 result.append(spelling[np.argmin(Jaccard_distance)])

             return result

         answer_ten()
```

```
Out[16]: ['cormus', 'incendiary', 'valid']
```

Question 11

For this recommender, your function should provide recommendations for the three default words provided above using the following distance metric:

Edit distance on the two words with transpositions. (https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance)

This function should return a list of length three: [`'cormulent_reccomendation'`, `'incendenece_reccomendation'`, `'validrate_reccomendation'`].

```
In [17]: def answer_eleven(entries=['cormulent', 'incendenece', 'validrate']):

             result = []

             for entry in entries:

                 spelling = [x for x in correct_spellings if x[0] == entry[0] and len(x) >= 5]

                 Damerau_Levenshtein_distance = [nltk.edit_distance(x, entry,transpositions=True) for x in spelling]

                 result.append(spelling[np.argmin(Damerau_Levenshtein_distance)])

             return result

         answer_eleven()
```

```
Out[17]: ['corpulent', 'intendence', 'validate']
```