*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) 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 [22]:
```
import nltk
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 'te
xt1'
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 [23]:
```
def example_one():

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

example_one()
```

Out[23]: 254989

Example 2

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

This function should return an integer.

In [24]:
```
def example_two():

    return len(set(nltk.word_tokenize(moby_raw))) # or alternatively len(set(t
ext1))

example_two()
```

Out[24]: 20755

Example 3

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

This function should return an integer.

In [25]:
```
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[25]: 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 [26]:
```
def answer_one():

    return len(set(moby_tokens))/ len(moby_tokens)

answer_one()
```

Out[26]: 0.08139566804842562

Question 2

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

This function should return a float.

In [27]:
```
def answer_two():
    #moby_tokens_lower = [w.lower() for w in moby_tokens]
    #moby_tokens_whale = [w for w in moby_tokens_lower if w == 'whale']
    #len(moby_tokens_whale) *100 / len(moby_tokens)
    moby_tokens_whale = [w for w in moby_tokens if w == 'whale' or w == 'Whal
e']


    return len(moby_tokens_whale) *100 / len(moby_tokens)

answer_two()
```

Out[27]:
```
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 [28]:
```
def answer_three():
    freq_dist = nltk.FreqDist(moby_tokens)



    return freq_dist.most_common(20)

answer_three()
```

Out[28]:
```
[(',', 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 a sorted list of the tokens that match the above constraints. To sort your list, use `sorted()`

In [29]:
```
def answer_four():
    freq_dist = nltk.FreqDist(moby_tokens)
    moby_df = pd.DataFrame(list(freq_dist.items()), columns = ['key', 'values'
])
    values_df = moby_df[moby_df['values'] > 150]
    key_df = values_df[values_df.key.str.len() > 5].sort_values('key')



    return list(key_df['key'])

answer_four()
```

Out[29]:
```
['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 [30]:
```
def answer_five():
    freq_dist = nltk.FreqDist(moby_tokens)
    moby_df = pd.DataFrame(list(freq_dist.items()), columns = ['key', 'values'
])
    moby_df['length'] = moby_df.key.str.len()
    longest = moby_df[moby_df['length'] == moby_df['length'].max()]
    (longest.iloc[0]['key'],longest.iloc[0]['length'] )


    return (longest.iloc[0]['key'],longest.iloc[0]['length'] )

answer_five()
```

Out[30]:
```
("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 [31]:
```
def answer_six():
    freq_dist = nltk.FreqDist(moby_tokens)
    moby_df = pd.DataFrame(list(freq_dist.items()), columns = ['key', 'values'
])
    gt2000 = moby_df[moby_df['values'] > 2000]
    gt2000 = gt2000.sort_values('key').tail(-3)
    gt2000 = gt2000.sort_values('values', ascending=False)


    return list(zip(gt2000['values'], gt2000['key']))

answer_six()
```

Out[31]:
```
[(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 [32]:
```
def answer_seven():
    moby_sent = nltk.sent_tokenize(moby_raw)
    moby_sent_df = pd.DataFrame(moby_sent, columns = ['sent'])
    sent_lengths = [len(nltk.word_tokenize(s)) for s in moby_sent]
    avg = sum(sent_lengths) / float(len(sent_lengths))


    return avg

answer_seven()
```

Out[32]:
```
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 [33]:
```
def answer_eight():
    pos = nltk.pos_tag(moby_tokens)
    pos_df = pd.DataFrame(pos)
    pos_freq = pos_df[1].value_counts()
    pos_freq = pos_freq.sort_values(ascending=False)
    pos_freq_top = pos_freq.head()



    return list(zip(pos_freq_top.index, pos_freq_top))
answer_eight()
```

Out[33]:
```
[('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 [34]:
```
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 [35]:
```
def answer_nine(entries=['cormulent', 'incendenece', 'validrate']):
    import nltk
    from nltk.corpus import words
    correct_spellings = words.words()
    from nltk.metrics.distance import jaccard_distance
    from nltk.util import ngrams

    answers = []
    for entry in entries:

        spellings = []
        for w in correct_spellings:
            if w.startswith(entry[0]):
                spellings.append(w)

        distance = []
        for word in spellings:
            dist = (nltk.jaccard_distance(set(ngrams(entry,3)), set(ngrams(wor
d,3))), word)
            distance.append(dist)
        answers.append(min(distance)[1])


    return answers

answer_nine()
```

Out[35]: `['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 [36]:
```
def answer_ten(entries=['cormulent', 'incendenece', 'validrate']):

    import nltk
    from nltk.corpus import words
    correct_spellings = words.words()
    from nltk.metrics.distance import jaccard_distance
    from nltk.util import ngrams

    answers = []
    for entry in entries:

        spellings = []
        for w in correct_spellings:
            if w.startswith(entry[0]):
                spellings.append(w)

        distance = []
        for word in spellings:
            dist = (nltk.jaccard_distance(set(ngrams(entry,4)), set(ngrams(wor
d,4))), word)
            distance.append(dist)
        answers.append(min(distance)[1])


    return answers

answer_ten()
```

Out[36]: ['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 [37]:
```
def answer_eleven(entries=['cormulent', 'incendenece', 'validrate']):
    import nltk
    from nltk.corpus import words
    correct_spellings = words.words()

    answers = []
    for entry in entries:

        spellings = []
        for w in correct_spellings:
            if w.startswith(entry[0]):
                spellings.append(w)

        distance = []
        for word in spellings:
            dist = (nltk.edit_distance(entry, word), word)
            distance.append(dist)
        answers.append(min(distance)[1])


    return answers

answer_eleven()
```

Out[37]:
```
['corpulent', 'intendence', 'validate']
```