

Chapter 1

You should install NLTK, import NITK and download the NLTK Book Collection at the beginning.

```
In [2]: ▶ # nltk.download()  
        ## -- I have it already installed on my system so I will just import it but this is the code to install
```

```
In [9]: ▶ import nltk
```

Question 1.1 Find the collocations in text2 from NLTK Book Collection

Defintion: A collocation is a sequence of words that occur together unusually often. Thus red wine is a collocation, whereas the wine is not. A characteristic of collocations is that they are resistant to substitution with words that have similar senses; for example, maroon wine sounds definitely odd.

```
In [10]:  from nltk.book import *
```

```
*** 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 [3]:  text2
```

```
Out[3]:  <Text: Sense and Sensibility by Jane Austen 1811>
```

```
In [11]:  print('; '.join(text2.collocation_list()))
```

```
# This code is to get the list of collocations from the text 5 from nltk library and we can see the output
```

```
Colonel Brandon; Sir John; Lady Middleton; Miss Dashwood; every thing; thousand pounds; dare say; Miss Stee
les; said Elinor; Miss Steele; every body; John Dashwood; great deal; Harley Street; Berkeley Street; Miss
Dashwoods; young man; Combe Magna; every day; next morning
```

Explanation: We use the `collocation_list()` function on any text for which we wish to have the collocations.

Question 1.2 Review the discussion of conditionals in 4. Find all words in the Chat Corpus (text5) ending with the letter l. Show them in alphabetical order.

Method 1: I am running it for all texts while caring about the lower or upper case

```
In [5]: ▶ sorted(w for w in set(text5) if w.endswith('l'))
```

```
'al',  
'alcohol',  
'all',  
'anal',  
'angel',  
'animal',  
'anygirl',  
'april',  
  
'asl',  
'astral',  
'atl',  
'bacl',  
'bagel',  
'ball',  
'barrel',  
'bbl',  
'beautiful',  
'bell',  
'betrayal',  
...
```

```
In [6]: ▶ len(sorted(w for w in set(text5) if w.endswith('l')))
```

Out[6]: 203

Method 2: I am running it for all texts without considering the cases now.

```
In [7]: sorted(set(w.lower() for w in text5 if w.endswith('l')))
```

```
Out[7]: ['ll',  
        'al',  
        'alcohol',  
        'all',  
        'anal',  
        'angel',  
        'animal',  
        'anygirl',  
        'april',  
        'asl',  
        'astral',  
        'atl',  
        'bacl',  
        'bagel',  
        'ball',  
        'barrel',  
        'bbl',  
        'beautiful',  
        'bell',  
        ..
```

```
In [8]: len(sorted(set((w.lower() for w in text5 if w.endswith('l')))))
```

```
Out[8]: 181
```

Explanation: We run a for loop on text5 for unique words which ends with the letter "l" and then sort them. The default order is ascending order. I also checked for how many such words are there and I found a total of 203 such words with taking in account the cases and 181 without cases.

Question 1.3 What is the difference between the following two lines of codes? Which one will give a larger value? Will this be the case for other texts?

1. `sorted(set(w.lower() for w in text2))`

2. `sorted(w.lower() for w in set(text2))`

Explanation: First let's understand what does set method do? It basically returns only the unique values. For example: I have a list as `a = [1,2,3,3,4,4,4,4,5]`. If I run `set(a)` then it will give me `-> (1,2,3,4,5)`

In the first case we are looking for all the unique words which have been converted into lower case from `text2`. In the second case we are converting into lower case all the unique words in `text2`.

The first one will have less because it first converts and then look for uniqueness while the second one first look for uniqueness and will include some words which are same but have different capitalizations and then converting them into lower case will lead to more records. Let's check the lengths for each code and see if our assumption is correct.

```
In [9]: ▶ len(sorted(set(w.lower() for w in text2)))
```

```
Out[9]: 6403
```

```
In [10]: ▶ len(sorted(w.lower() for w in set(text2)))
```

```
Out[10]: 6833
```

As assumed earlier we ran the code and included it inside the `len()` and found that second one give a larger value. The same thing will hold for any other text unless we see a special case where all words are in lower case such that the unique values with or without `lower()` method remains same.

Question 1.4 Find all the Twelve-letter words in the Chat Corpus (`text2`). With the help of a frequency distribution (`FreqDist`), show these words in decreasing order of frequency.

There is a confusion to whether run it for chat corpus which is `text 5` or `text 2`? So I run it for both

For Text 5: Chat Corpus

```
In [9]: ▶ sorted(set([w for w in text5 if w.isalpha() and len(w) == 12]))
```

```
Out[9]: ['Blooooooooood',  
         'Christianity',  
         'Connecticut',  
         'Considerably',  
         'Constitution',  
         'Fergalicious',  
         'alternatives',  
         'bachelorette',  
         'butterscotch',  
         'catastrophic',  
         'christianity',  
         'constituents',  
         'construction',  
         'conversation',  
         'denomination',  
         'discriminate',  
         'disappointed',  
         'entertaining',  
         'freeeezinggg',  
         'hahahahahaha',  
         'heartbreaker',  
         'heyyyyyyyyyy',  
         'hhaaaaatttee',  
         'hugssssssssss',  
         'ihavehotnips',  
         'interruption',  
         'masturbating',  
         'multitasking',  
         'necromancers',  
         'neighborhood',  
         'oooooooooooo',  
         'outrageously',  
         'passionately',  
         'periodically',  
         'psychologist',  
         'registration',  
         'slkfjsldkfjs',  
         'subscription',  
         'sweeeeeeeeet',
```

```
'thanksgiving',  
'tooooooooooooo',  
'yummylicious']
```

```
In [10]: ► len(sorted(set([w for w in text5 if w.isalpha() and len(w) == 12])))
```

```
Out[10]: 42
```

```
In [11]: ▶ chat_corpus = sorted([w for w in text5 if w.isalpha() and len(w) == 12])
FreqDist(chat_corpus).most_common()
```

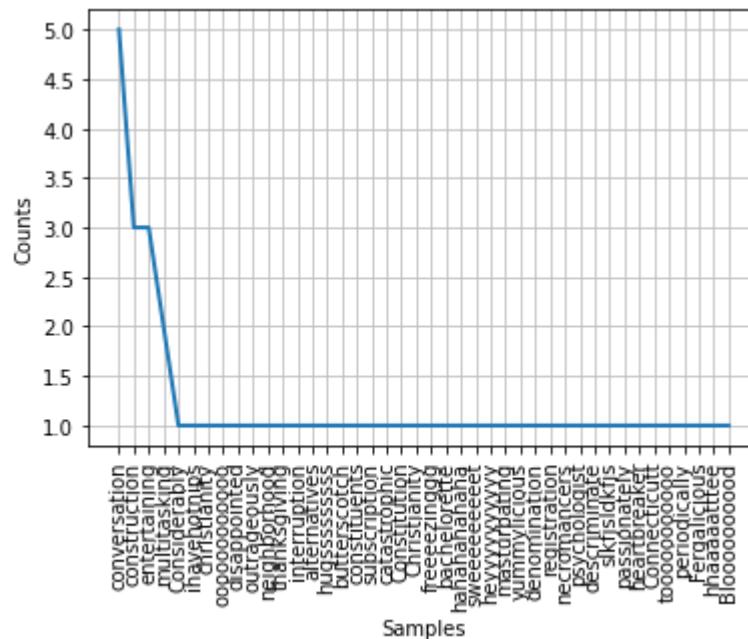
```
Out[11]: [('conversation', 5),
('construction', 3),
('entertaining', 3),
('multitasking', 2),
('Blooooooooood', 1),
('Christianity', 1),
('Connecticut', 1),
('Considerably', 1),
('Constitution', 1),
('Fergalicious', 1),
('alternatives', 1),
('bachelorette', 1),
('butterscotch', 1),
('catastrophic', 1),
('christianity', 1),
('constituents', 1),
('denomination', 1),
('discriminate', 1),
('disappointed', 1),
('freeeezinggg', 1),
('hahahahahaha', 1),
('heartbreaker', 1),
('heyyyyyyyyyy', 1),
('hhaaaaatttee', 1),
('hugsssssssss', 1),
('ihavehotnips', 1),
('interruption', 1),
('masturbating', 1),
('necromancers', 1),
('neighborhood', 1),
('oooooooooooo', 1),
('outrageously', 1),
('passionately', 1),
('periodically', 1),
('psychologist', 1),
('registration', 1),
('slkfjsldkfjs', 1),
('subscription', 1),
```



```
('sweetest', 1),
('thanksgiving', 1),
('tooooooooooooo', 1),
('yummylicious', 1)]
```

```
In [13]: ▶ df = FreqDist(text5)
words = sorted([w for w in text5 if w.isalpha() and len(w) == 12])
for i in [w for w in df]:
    if i not in words:
        df.pop(i)
print(df)
df.plot()
```

<FreqDist with 42 samples and 51 outcomes>



Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1b2a4def9e8>

For Text 5: Sense and Sensibility by Jane Austen 1811

```
In [14]: ► [w for w in text2 if w.isalpha() and len(w) == 12]
```

```
Out[14]: ['acquaintance',  
          'cheerfulness',  
          'articulation',  
          'economically',  
          'considerable',  
          'considerable',  
          'successively',  
          'affectionate',  
          'wretchedness',  
          'cheerfulness',  
          'relationship',  
          'expectations',  
          'disagreeable',  
          'independence',  
          'inconvenient',  
          'occasionally',  
          'unreasonable',  
          'housekeeping',  
          'irresistible',  
          ]
```

```
In [15]: ► len(sorted(set([w for w in text2 if len(w) == 12])))
```

```
Out[15]: 205
```

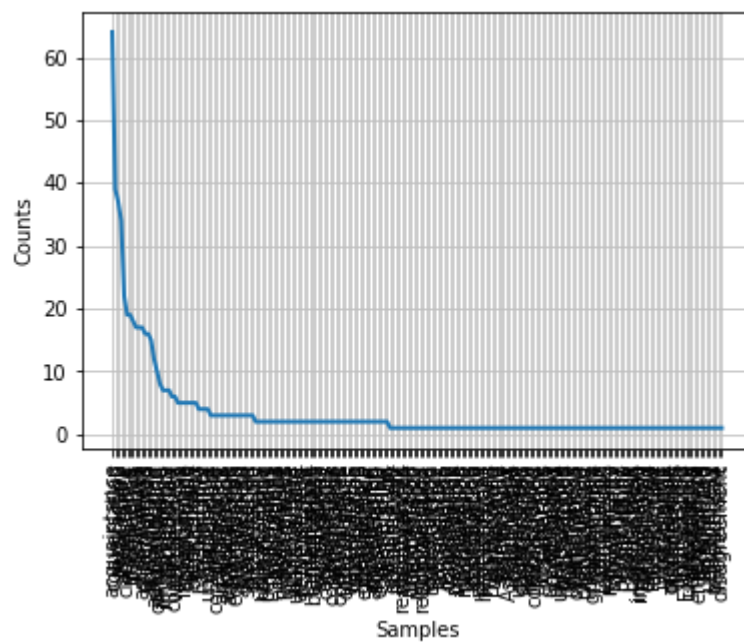
```
In [16]: ► text2_words = [w for w in text2 if len(w) == 12]
```

```
In [17]: ► fdist_text2 = FreqDist(text2_words)
```



```
In [19]: ▶ df = FreqDist(text2)
words = sorted([w for w in text2 if w.isalpha() and len(w) == 12])
for i in [w for w in df]:
    if i not in words:
        df.pop(i)
print(df)
df.plot()
```

<FreqDist with 205 samples and 711 outcomes>



Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1b2a4ef1a20>

Question 1.5 Review the discussion of looping with conditions in Section 4 in Chapter 1. Use a combination of for and if statements to loop over the tokens of text2 and print all the distinct punctuation, one per line.

Method 1: I use the punctuation class from string package to find all the punctuations in the text 2

```
In [20]:  from string import punctuation as punc
```

```
In [21]:  set(w for w in text2 if w in punc)
```

```
Out[21]: {'!',  
          '!',  
          '!',  
          '!',  
          '&',  
          '"',  
          '(',  
          ')',  
          ',',  
          '-',  
          '.',  
          ':',  
          ';',  
          '>',  
          '?',  
          '[',  
          '']
```

```
In [22]:  len(set(w for w in text2 if w in punc))
```

Out[22]: 17

Method 2: I use to find all varieties of punctuation by selecting anything that is not alpha numeric.

```
In [23]: text2_punc = [w for w in text2 if not w.isalnum()]
```

```
In [24]: text2_punc
```

[illegible]

```
In [25]: ▶ len(text2_punc)
```

Out[25]: 20789

Explanation: I call the punctuation method from string class to get all the punctuations by running a for loop on text 2 with an if condition for punctuations and then pass this entire code inside set method for unique values. There are 17 different punctuations in text 2.

Chapter 2

Question 2.1 Use Gutenberg Corpus Module to explore 'shakespeare-caesar.txt'.

- How many word tokens does this book have?
- How many word types?
- What is the lexical richness?

```
In [13]: ➤ from nltk.corpus import gutenberg
```

```
In [18]: ➤ nltk.corpus.gutenberg.fileids()
```

```
Out[18]: ['austen-emma.txt',  
          'austen-persuasion.txt',  
          'austen-sense.txt',  
          'bible-kjv.txt',  
          'blake-poems.txt',  
          'bryant-stories.txt',  
          'burgess-busterbrown.txt',  
          'carroll-alice.txt',  
          'chesterton-ball.txt',  
          'chesterton-brown.txt',  
          'chesterton-thursday.txt',  
          'edgeworth-parents.txt',  
          'melville-moby_dick.txt',  
          'milton-paradise.txt',  
          'shakespeare-caesar.txt',  
          'shakespeare-hamlet.txt',  
          'shakespeare-macbeth.txt',  
          'whitman-leaves.txt']
```

```
In [19]: ➤ caesar = gutenberg.words('shakespeare-caesar.txt')
```

```
In [67]: ▶ print("Number of word tokens = ", len(caesar))
          print("Number of word types = ", len(set(caesar)))
          print("Lexical richness = ", round(len(set(caesar))/len(caesar)*100,2), "%")
```

```
Number of word tokens = 25833
Number of word types = 3560
Lexical richness = 13.78 %
```

Explanation: I call len() method to get total word tokens and len() with set() method to get word types. When I take the ratio of these two, I get the lexical richness of our file.

Question 2.2 Explore the Section "Hobbies" in Brown Corpus. Count the number of wh words, such as what, when, where, who and why. Please make sure we are not double-counting words like "This" and "this", which differ only in capitalization.

```
In [14]: ▶ from nltk.corpus import brown
```

```
In [15]: ▶ hobbies_text = brown.words(categories='hobbies')
```



```
In [16]: sorted(set(wh.lower() for wh in hobbies_text if wh.lower().startswith('wh')))
```

```
Out[16]: ['whaddya',  
          'whaling',  
          'what',  
          "what's",  
          'whatever',  
          'wheel',  
          'wheeled',  
          'wheels',  
          'when',  
          'whenever',  
          'where',  
          'whereas',  
          'whereby',  
          'wherein',  
          'wherever',  
          'whether',  
          'which',  
          'while',  
          'whip',  
          'whipped',  
          'whippet',  
          'whips',  
          'whirlwind',  
          'whiskey',  
          'white',  
          'whitetail',  
          'who',  
          'whole',  
          'whole-house',  
          'wholesale',  
          'wholly',  
          'whom',  
          'whose',  
          'why']
```

```
In [17]: ▶ len(sorted(set(wh.lower() for wh in hobbies_text if wh.lower().startswith('wh'))))
```

```
Out[17]: 34
```

```
In [18]: hobbies_text_cw = sorted(wh.lower() for wh in hobbies_text if wh.lower().startswith('wh'))
sorted(FreqDist(hobbies_text_cw).most_common())
```

```
Out[18]: [('whaddya', 1),
('whaling', 1),
('what', 108),
("what's", 1),
('whatever', 6),
('wheel', 4),
('wheeled', 2),
('wheels', 3),
('when', 164),
('whenever', 2),
('where', 77),
('whereas', 2),
('whereby', 1),
('wherein', 1),
('wherever', 3),
('whether', 14),
('which', 253),
('while', 38),
('whip', 1),
('whipped', 2),
('whippet', 1),
('whips', 1),
('whirlwind', 1),
('whiskey', 1),
('white', 15),
('whitetail', 2),
('who', 104),
('whole', 20),
('whole-house', 1),
('wholesale', 1),
('wholly', 2),
('whom', 2),
('whose', 14),
('why', 17)]
```

Explanation: Call the Brown Corpus and choose the 'hobbies' category. Use the FreqDist() method to call all the words in the text from the selected category. Ensure we have all the words converted to lower to avoid duplicates that may arise due to capitalization. Then run a loop for the words which starts with "wh". Use a for loop to run this array over the frequency distribution method to get the unique

counts for each of those "wh" words.

Question 2.3 Explore movie reviews corpus which contains 2k movie reviews with sentiment polarity classification (positive or negative reviews). Please find out the 20 most common words in the negative reviews and positive reviews. Please get rid of stop words, number and punctuations from reviews. Please make sure we are not double-counting words like “This” and “this”, which differ only in capitalization.

Hint:

- from nltk.corpus import movie_reviews
- from nltk.corpus import stopwords

```
In [7]:  from nltk.corpus import movie_reviews
        from nltk.corpus import stopwords
        from nltk import FreqDist
```

```
In [128]: len(movie_reviews.words()) # Prints total number of words in 'movie_reviews'
```

```
Out[128]: 1583820
```

```
In [3]:  movie_reviews.categories()
```

```
Out[3]: ['neg', 'pos']
```

```
In [133]: documents = []

for category in movie_reviews.categories():
    for fileid in movie_reviews.fileids(category):
        documents.append((movie_reviews.words(fileid), category))

print (len(documents))
```

2000

```
In [113]: len(movie_reviews.fileids("neg"))
```


Out[113]: 1000

```
In [114]: len(movie_reviews.fileids("pos"))
```

Out[114]: 1000

```
In [136]: text = movie_reviews.words()
```

Method 1: For both postive and negative words together

```
In [137]:  '''  
def most_freq_words(text):  
    fdist = FreqDist(w.lower() for w in text if w.isalpha() and w.lower() not in stopwords.words('english'))  
    return fdist.most_common(20)  
  
most_freq_words(text)  
'''
```

```
Out[137]: [('film', 9517),  
            ('one', 5852),  
            ('movie', 5771),  
            ('like', 3690),  
            ('even', 2565),  
            ('good', 2411),  
            ('time', 2411),  
            ('story', 2169),  
            ('would', 2109),  
            ('much', 2049),  
            ('character', 2020),  
            ('also', 1967),  
            ('get', 1949),  
            ('two', 1911),  
            ('well', 1906),  
            ('characters', 1859),  
            ('first', 1836),  
            ('see', 1749),  
            ('way', 1693),  
            ('make', 1642)]
```

Method 2: For both positive and negative words separately

```
In [23]: ► neg_w = []
for fileid in movie_reviews.fileids('neg'):
    check = movie_reviews.words(fileid)
    for w in check:
        if w.lower() not in stopwords and w.isalpha():
            neg_w.append(w.lower())
fdist_neg_w = FreqDist(neg_w)

pos_w = []
for fileid in movie_reviews.fileids('pos'):
    check = movie_reviews.words(fileid)
    for w in check:
        if w.lower() not in stopwords and w.isalpha():
            pos_w.append(w.lower())
fdist_pos_w = FreqDist(pos_w)
```

```
In [24]: ▶ fdist_neg_w.most_common(20)
```

```
Out[24]: [('film', 4287),  
          ('movie', 3246),  
          ('one', 2800),  
          ('like', 1888),  
          ('even', 1386),  
          ('time', 1168),  
          ('good', 1163),  
          ('would', 1090),  
          ('get', 1052),  
          ('bad', 1034),  
          ('much', 1011),  
          ('character', 942),  
          ('story', 923),  
          ('plot', 917),  
          ('two', 912),  
          ('characters', 873),  
          ('make', 851),  
          ('first', 832),  
          ('could', 791),  
          ('see', 784)]
```



```
In [25]: ▶ fdist_pos_w.most_common(20)
```

```
Out[25]: [('film', 5230),
          ('one', 3052),
          ('movie', 2525),
          ('like', 1802),
          ('good', 1248),
          ('story', 1246),
          ('time', 1243),
          ('also', 1200),
          ('even', 1179),
          ('well', 1123),
          ('character', 1078),
          ('life', 1057),
          ('much', 1038),
          ('would', 1019),
          ('first', 1004),
          ('two', 999),
          ('characters', 986),
          ('see', 965),
          ('way', 929),
          ('get', 897)]
```

```
In [21]: ▶ documents = []

for category in movie_reviews.categories():
    for fileid in movie_reviews.fileids("pos"):
        documents.append((movie_reviews.words(fileid), category))

print (len(documents))
```

```
2000
```

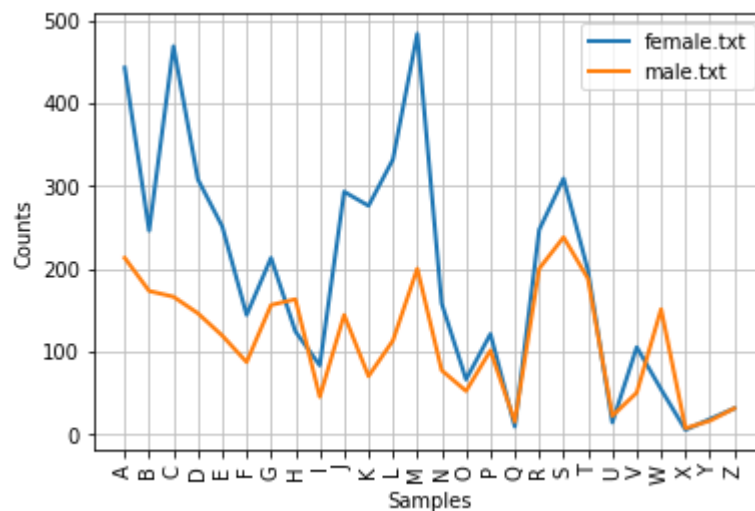
Explanation: We call the movie_reviews from nltk corpus and then check for the categories and we found that there are exactly 1000 reviews for both negative and positive types.

We then write a function that can find the frequency for the top 20 words. The conditions we will need inside the `FreqDist()` method is that the words must be lower case and are alphabets only which are not stopwords. The flow of which can be seen from the above code. Then use the most common method to pass a number as an argument which is basically the number of words we wish to run this function for.

Question 2.4 Explore Names Corpus, which initial letters are more frequent for males vs. females? ((Hint: Plot of a Conditional Frequency Distribution will be useful for answering this question))

```
In [88]:  from nltk.corpus import names
```

```
In [89]:  cfd = nltk.ConditionalFreqDist(
            (fileid, name[0])
            for fileid in names.fileids()
            for name in names.words(fileid))
cfd.plot()
```



```
Out[89]: <matplotlib.axes._subplots.AxesSubplot at 0x22b5523f668>
```

Explanation: From nltk corpus import the names package. Then pass the data into the conditional frequency distribution method to get the field ids with their counts where each field id is nothing but the 26 letters in the English language. We then plot it to have the records for male and female represented for each of those field ids.

Question 2.5 Use the Gutenberg Corpus module to explore austen-persuasion.txt. Write a function to find out the 20 most frequent occurring words of this text file that are not stopwords. Please get rid of numbers and punctuations. Please make sure we are not double-counting words like “This” and “this”, which differ only in capitalization.

```
In [100]: ▶ text = gutenbergs.words('austen-persuasion.txt')
```

```
In [102]: ▶ def most_freq_words(text):
            fdist = FreqDist(w.lower() for w in text if w.isalpha() and w.lower() not in stopwords.words('english'))
            return fdist.most_common(20)

            most_freq_words(text)
```

```
Out[102]: [('anne', 497),
            ('could', 451),
            ('would', 355),
            ('captain', 303),
            ('mrs', 291),
            ('elliot', 289),
            ('mr', 256),
            ('one', 238),
            ('must', 228),
            ('wentworth', 218),
            ('lady', 216),
            ('much', 205),
            ('good', 187),
            ('little', 176),
            ('said', 173),
            ('charles', 166),
            ('might', 166),
            ('well', 163),
            ('never', 155),
            ('time', 152)]
```

Explanation: Call the file mentioned in the question and store the words in a variable. Then write a function that can find the frequency of the top 20 words. The conditions we will need inside the FreqDist() method is that the words must be lower case and are alphabets only which are not stopwords. The flow of which can be seen from the above code. Then use the most common method to pass a number as an argument which is basically the number of words we wish to run this function for.

Question 2.6 Use one of the path similarity measures to score the similarity of each of the following pairs of words. Rank the pairs in order of decreasing similarity: car-automobile, journey-voyage, boy-lad, coast-shore, midday-noon, furnace-stove, food-fruit, bird-cock, bird-crane, tool-implement, journey-car, cemetery-woodland, food-rooster, coast-hill, forest-graveyard, shore-woodland, coast-forest, lad-wizard, chord-smile, glass-magician, noon-string.

```
In [28]: ▶ from nltk.corpus import wordnet as wn
```

```
In [29]: ▶ pairs = [('car', 'automobile'), ('journey', 'voyage'), ('boy', 'lad'), ('coast', 'shore'), ('midday', 'noon'),  
                    ('furnace', 'stove'), ('food', 'fruit'), ('bird', 'cock'), ('bird', 'crane'), ('tool', 'implement'),  
                    ('journey', 'car'), ('cemetery', 'woodland'), ('food', 'rooster'), ('coast', 'hill'), ('forest', 'shore'),  
                    ('shore', 'woodland'), ('coast', 'forest'), ('lad', 'wizard'), ('chord', 'smile'),  
                    ('glass', 'magician'), ('noon', 'string')]
```

```
In [34]: ▶ from operator import itemgetter
```

```
In [33]: ▶ path = []
for w1, w2 in pairs:
    path.append((w1, w2, wn.path_similarity(wn.synsets(w1)[0], wn.synsets(w2)[0])))
sorted(path, key=itemgetter(2), reverse=True)
```

```
Out[33]: [('car', 'automobile', 1.0),
('midday', 'noon', 1.0),
('coast', 'shore', 0.5),
('tool', 'implement', 0.5),
('boy', 'lad', 0.3333333333333333),
('journey', 'voyage', 0.25),
('coast', 'hill', 0.2),
('shore', 'woodland', 0.2),
('lad', 'wizard', 0.2),
('bird', 'crane', 0.1111111111111111),
('cemetery', 'woodland', 0.1111111111111111),
('glass', 'magician', 0.1111111111111111),
('food', 'fruit', 0.09090909090909091),
('coast', 'forest', 0.09090909090909091),
('chord', 'smile', 0.09090909090909091),
('furnace', 'stove', 0.07692307692307693),
('forest', 'graveyard', 0.07142857142857142),
('bird', 'cock', 0.0625),
('food', 'rooster', 0.0625),
('noon', 'string', 0.058823529411764705),
('journey', 'car', 0.05)]
```

Explanation: I create an array for each pair and then we compare the path similarity for the pairs based on synsets for word 1 on word 2 and then put the results in decreasing order using sorted method.