IST664 - Homework 1

Student name: Benjamin Tisinger

Originality assertion: By adding my name above, I assert that all of the text and comments in this file are my original work (except for template items written by the instructor). All of the code in this file is my work, except where I give credit to another source.

Note: You may freely use code from Labs for this class without the need for attributions.

Also note: DO NOT under any circumstances borrow the code of another student. You should feel free to help each other with coaching and suggestions, but do not share code.

```
# In this code, we import the NLTK, download the Gutenberg texts, extract a
# list of file identifiers from the downloaded material.

import nltk # Bring in the NLP toolkit
nltk.download('gutenberg') # Then import the Gutenberg library, which has books
nltk.corpus.gutenberg.fileids()
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
['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']
```

```
# Choose a file index using the list above, except for 0 and 1, where the first
# letters of the file name are closest to the first letters of your last name.

my_file_id = 7 # Change to any value between 2 and 17

my_file = nltk.corpus.gutenberg.fileids()[my_file_id] # Extract the file

my_text = nltk.corpus.gutenberg.raw(my_file) # Here's the raw text
print(my_text)
```

```
'You're a very poor speaker,' said the King.

Here one of the guinea-pigs cheered, and was immediately suppressed by
the officers of the court. (As that is rather a hard word, I will just
explain to you how it was done. They had a large canvas bag, which tied
up at the mouth with strings: into this they slipped the guinea-pig,
head first, and then sat upon it.)

'I'm glad I've seen that done,' thought Alice. 'I've so often read
in the newspapers, at the end of trials, "There was some attempts
at applause, which was immediately suppressed by the officers of the
court," and I never understood what it meant till now.'

'If that's all you know about it, you may stand down,' continued the
King.

'I can't go no lower,' said the Hatter: 'I'm on the floor, as it is.'

'Then you may SIT down,' the King replied.

Here the other guinea-pig cheered, and was suppressed.

'Come, that finished the guinea-pigs!' thought Alice. 'Now we shall get
on better.'

'I'd rather finish my tea,' said the Hatter, with an anxious look at the
Queen, who was reading the list of singers.

'You may go,' said the King, and the Hatter hurriedly left the court,
without even waiting to put his shoes on.
```

Part 1

Make sure that you have (at least) one block of code for each of the tasks.

1. Tokenize your book and print how many tokens it contains
2. Remove stop words and punctuation; lowercase all remaining tokens; make sure to use this result in subsequent operations for Part 1.
3. Recalculate and display the total number of remaining tokens (i.e., after removal of stop words and punctuation)
4. Find all tokens with a length (in characters) greater than eight; print at least one example from this list and print the length of this list
5. Conduct a frequency analysis of the tokens and store it in an appropriate data structure; display the type of that data structure
6. Display the 20 most frequently occurring tokens and how often they occur
7. Obtain a list of the unique set of tokens and print the length of this list

```
#
# Begin tasks 1 - 7 here.
#
```

```
#Task 1
import nltk
nltk.download('punkt_tab')

len(my_text)
tokens = nltk.word_tokenize(my_text)
len(tokens)
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
33535
```

```
#Task 2
import string

nltk.download('stopwords')
nltk_stops = nltk.corpus.stopwords.words('english')

filtered_tokens = [word.lower() for word in tokens if word.lower() not in nltk_stops]
filtered_tokens = [word for word in filtered_tokens if word not in string.punctuation]

print(filtered_tokens)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
['alice', "'s", 'adventures', 'wonderland', 'lewis', 'carroll', '1865', 'chapter', 'rabbit-hole', 'alice', 'beginning', 'get', 'tired',
```

```
#Task 3
len(filtered_tokens)
```

```
13853
```

```
#Task 4
eight_tokens = [word for word in filtered_tokens if len(word) > 8]
print(eight_tokens)
print(len(eight_tokens))
```

```
['adventures', 'wonderland', 'rabbit-hole', 'beginning', 'conversations', 'conversation', 'considering', 'daisy-chain', 'remarkable', 'a
894
```

```
#Task 5
from collections import Counter
frequency = Counter(filtered_tokens)
type(frequency)
#Used Sources Lab 2 and https://www.phind.com/
```

```
collections.Counter
def __init__(self, iterable=None, /, **kwds)

/usr/lib/python3.11/collections/__init__.py
Dict subclass for counting hashable items.  Sometimes called a bag
or multiset.  Elements are stored as dictionary keys and their counts
are stored as dictionary values.

>>> c = Counter('abcdeabcdabcaba')  # count elements from a string
```

```
#Task 6
frequency.most_common(20)
```

```
[('said', 462),
 ('alice', 397),
 ('--', 264),
 ("n't", 217),
 ("'s", 201),
 ('little', 128),
 ('one', 99),
 ('would', 90),
 ('know', 88),
 ('could', 86),
 ('like', 85),
 ('went', 83),
 ('queen', 75),
 ('thought', 74),
 ("''", 73),
 ('time', 68),
 ('see', 67),
 ('king', 63),
 ("'m", 59),
 ('turtle', 59)]
```

```
#Task 7
unique_words = set(filtered_tokens)
print(len(unique_words))
```

```
2668
```

Part 2

A Regular Expression (RegEx), is a sequence of characters that forms a search pattern. RegEx can be used to check if a string contains the specified search pattern. You can use the re package to test whether a regular expression matches a specific string in Python. All of the following exercises should be done using regex expressions.

8. Go back to the “raw” text of your book and use a sentence tokenizer to divide the text into a list of sentences. Display the total number of sentences in the list. Perform the following parsing operations on a sample of sentences using RegEx patterns. For readability, I suggest that you do each of these items in a separate code block:
9. Divide the sentence into separate tokens based on whitespace.
10. Discard all punctuation characters.
11. Remove plural endings by discarding a trailing "s" on any token.

12. Remove "ed" endings (i.e., the past tense on a verb such as "framed").

It doesn't really matter how many sentences you process, just so long as you do enough to demonstrate that your Regex code works. Make sure that the output displayed in the notebook clearly demonstrates the success of your Regex code. Put in a concluding comment that discusses how well your Regex works, compared with a more full-featured stemmer (e.g., the Porter stemmer).

```
#
# Begin tasks 8 - 12 here.
#
```

```
#Task 8
my_file_id = 7
my_file = nltk.corpus.gutenberg.fileids()[my_file_id]
my_text = nltk.corpus.gutenberg.raw(my_file)

text_sentences = nltk.sent_tokenize(my_text)
print(len(text_sentences))
print(text_sentences)
```

```
1625
["[Alice's Adventures in Wonderland by Lewis Carroll 1865]\n\nCHAPTER I.", "Down the Rabbit-Hole\n\nAlice was beginning to get very tire
```

```
#Sentences
import random

sent1, sent2 = random.sample(text_sentences, 2)

print("Sentence 1:", sent1)
print("Sentence 2:", sent2)

#Used Sources Lab2 and https://www.phind.com/
```

```
Sentence 1: 'Hold your tongue, Ma!'
Sentence 2: 'Up, lazy
thing!'
```

```
#Task 9 -Sent1
import re
tokens1 = re.findall(r'\b\w+\b', sent1)
print(tokens1)
```

```
['Hold', 'your', 'tongue', 'Ma']
```

```
#Task 9 -Sent2
tokens2 = re.findall(r'\b\w+\b', sent2)
print(tokens2)
```

```
['Up', 'lazy', 'thing']
```

```
#Task 10 - Sent1
import string
filtered_tokens1 = [word for word in tokens1 if word not in string.punctuation]
print(filtered_tokens1)
```

```
['Hold', 'your', 'tongue', 'Ma']
```

```
#Task 10 - Sent2
filtered_tokens2 = [word for word in tokens2 if word not in string.punctuation]
print(filtered_tokens2)
```

```
['Up', 'lazy', 'thing']
```

```
#Task 11 - Sent1
filtered_tokens1 = [word.rstrip('s') for word in filtered_tokens1]
print(filtered_tokens1)
```

```
['Hold', 'your', 'tongue', 'Ma']
```

```
#Task 11 - Sent2
filtered_tokens2 = [word.rstrip('s') for word in filtered_tokens2]
print(filtered_tokens2)
```

```
['Up', 'lazy', 'thing']
```

```
#Task 12 - Sent1
filtered_tokens1 = [word.rstrip('ed') for word in filtered_tokens1]
print(filtered_tokens1)
```

```
['Hol', 'your', 'tongu', 'Ma']
```

```
#Task 12 - Sent2
filtered_tokens2 = [word.rstrip('ed') for word in filtered_tokens2]
print(filtered_tokens2)
```

```
['Up', 'lazy', 'thing']
```

```
#Comment About REGEX vs Porter
#REGEX in this situation is very basic and does not offer a robust outcome as the Porter Stemmer would. The REGEX we used is very basic
#such as Geese, Mice, Children etc... I would say REGEX is good for doing simple tasks or testing. If this was a pratical situtation for
#REGEX is for simple tasks and basic manipulation while Porter Stemmer is actually designed to reduce words to their basic stem or root.
```

```
#Just for Fun - Testing with Porter Stemmer
from nltk.stem import PorterStemmer

ps = PorterStemmer()
filtered_tokens2 = [ps.stem(word) for word in filtered_tokens2]
print(filtered_tokens2)

```

```
['up', 'lazi', 'thing']
```

```
#Just for Fun - Testing with Porter Stemmer
from nltk.stem import PorterStemmer

ps = PorterStemmer()
filtered_tokens1 = [ps.stem(word) for word in filtered_tokens1]
print(filtered_tokens1)
```

```
['hol', 'your', 'tongu', 'ma']
```