MSiA 414: Text Analytics Fall 2021
HW1

Louis-Charles Généreux
Github link:
https://github.com/MSIA/lgo4950_msia_text_analytics_2021/tree/homework1/HW1_submission

Question 1:

For this question, I analyze the performance of two common NLP packages: NLTK and Spacy. I will compare their performance on a set of different operations, using text from the first 20 chapters of Jane Austen's *Pride and Prejudice (199920 characters).* More specifically, I compare their execution speed across 10 different runs (averaging the execution time across the runs) for 4 operations:

- Word tokenization: Dividing text into individual words
- Sentence tokenization: Dividing text into sentences
- POS tagging: Grammatical tagging (assigning each individual word to its part of speech)
- Stemming: Identifying the root of a word (stripping forms of derivatives on a given word)

Across both libraries, tokenization is significantly faster than POS tagging and stemming (see results table below). For tokenization in NLTK, sentence tokenization is faster than word tokenization; the opposite is true with Spacy. After consulting Spacy documentation, I learned that this is because the underlying Spacy tokenization code first splits text into words, then reconstructs sentences (this explains why sentence tokenization is slower than word tokenization in Spacy).

	NLTK	Spacy
Word token	0.209777	0.056593
Sentence token	0.052850	0.291732
POS tagging	1.604599	2.055069
Stemming	1.565730	2.132924

Aside from speed and performance, I was also interested in the 'usability' of both packages. I enjoyed Spacy's customization, where users can customize the pipeline to speed up operations (by reducing the number of operations being computed at each call); the flip side is that for a same call, many attributes can be computed (https://spacy.io/usage/processing-pipelines). This differs from NLTK, where many calls would be required to gather stemming, tagging and tokenization. Another difference was that Spacy is object oriented (both the pipeline and outputs are objects), which allows for high customization.

Question 2 instructions for grader:

From the root of my submissions folder, run the following commands to see the input and output of my function and the proper unit tests.

> ***python question_2.py*** (running the py file where my two functions are written)
> ***pytest question_2_tests.py*** (formally running unit tests)

Question 2.1:

To test my function, I used the code snippet provided in the assignment as a string (see below). Here is my output.

```
(py3) [21:58:00] louisgenereux:submission git:(main) $ python question_2.py
Sample text:
def foo():
        print("foo was called")

def bar(x):
        print("bar was called with an argument %s"%x)

Output:
['foo', 'bar']

--------------------------------------------------
```

Question 2.2:

To test my function, I used a string that includes multiple variations of date formats:

1. mm/dd/yyyy with 4 year digits (**04/12/2019**)
2. mm/dd/yyyy with 2 year digits (**04/12/19**)
3. mm-dd-yyyy with 4 year digits (**04-12-2019**)
4. mm-dd-yyyy with 2 year digits (**04-12-19**)
5. mm.dd.yyyy with 4 year digits (**04.12.2019**)
6. mm.dd.yyyy with 2 year digits (**04.12.19**)
7. Uppercase month string dd yyyy (**April 12 2019**)
8. Lowercase month string dd yyyy (**april 12 2019**)
9. Uppercase month string dd (with th) yyyy (**April 12th 2019**)
10. Lowercase month string dd (with th) yyyy (**april 12th 2019**)
11. Uppercase month string dd yy (two digit year only) (**April 12 19**)
12. Lowercase month string dd yy (two digit year only) (**april 12 19**)
13. Uppercase month string dd (with th) yy (two digit year only) (**April 12th 19**)
14. Lowercase month string dd (with th) yy (two digit year only) (**april 12th 19**)

Output from my code:

```
--------------------------------------------------
Sample text:
Random text here 04/12/2019 and here. Hello...  qwerty: 04/12/19!  Random text here 04-12-2019 ...
04-12-19... Random text here 04.12.2019! Random text here 04.12.19 April 12 2019 is a great date! S
o is april 12 2019! Another sentence including April 12th 2019 and april 12th 2019, along with... A
pril 12 19. This is starting to be a long list, but is almost over: april 12 19, April 12th 19 and
april 12th 19 are finally the last of 14 dates  that I have tried

Output:
['04/12/2019', '04/12/19', '04-12-2019', '04-12-19', '04.12.2019', '04.12.19', 'April 12th 2019', '
april 12th 2019', 'April 12th 19', 'april 12th 19', 'April 12 2019', 'april 12 2019', 'April 12 19'
, 'april 12 19']

--------------------------------------------------
```

Unit tests for question 2:

Unit tests for both functions run successfully

```
(py3) [22:25:40] louisgenereux:submission git:(main) $ pytest question_2_tests.py
============================================================================= test session starts =
platform darwin -- Python 3.7.9, pytest-5.4.2, py-1.10.0, pluggy-0.13.1
rootdir: /Users/louisgenereux/Desktop/Term 4/Text_analytics/HW1/submission
collected 2 items

question_2_tests.py ..

============================================================================== 2 passed in 0.32s ==
```