Python Project Singers

Assignment Overview

This assignment will give you more experience on the use of dictionaries and sets.

The goal of this project is to investigate singers and songs to calculate vocabulary of singers and average number of distinct words per songs. Also, the aim is to develop a simple search engine to search through songs by using keywords.

Given song data in a comma separated values (CSV) and stop words in a text file (txt), which is a file containing the words we wish to ignore, extract and store the necessary data you need to answer the questions about singers and songs. Use dictionaries to store singers, song names and sets to store stop words and lyrics after splitting and removing the stop words.

Assignment Background

You are hired as a junior software developer for YourTunes Company. YourTunes is a music service which is used by millions of people. However, the users always complain about the search quality of the application. Your manager wants you to develop a search engine so that users could search lyrics by words. Also, you are expected to create a brief report about the singers which the company has a copyright agreement. You are provided a detailed specification document for these tasks. You want to solve these tasks by using efficient algorithms and you remember how dictionaries and sets operate and you want to use these data structures for this project. You want to start working as soon as possible to impress your manager.

Project Description

You are given proj09.py which contains the skeleton of all the functions you are required to implement. You can define more functions if you wish. The functions in proj09.py will be individually tested in grading to ensure their proper function. Remember, function parameters can be named anything you want – so feel free to change their names so they make more sense to you, but do not change the names of the functions themselves. Also, the order of the parameters is important and should not be changed.

You must implement the following functions:

`open_file(message) -> fp`

This function prompts the user to enter a filename displaying the message provided. The program will try to open a file. It should have a try-except statement. An error message should be shown if the file cannot be opened. This function will loop until it receives proper input and successfully opens the file. It returns a file object.

`read_stopwords(fp) -> set`
This function receives a file pointer as a parameter (such as returned from open_file(message)) and reads the file line by line to create a set of stop words. All of the words should be converted to lower case.

`validate_word(word, stopwords) -> Boolean`
This function receives a string and a set as parameters . If the given word is in the stop word set or it has any digit or punctuation, the function returns False. Otherwise, it returns True. Hint: the string method isalpha() may prove useful here. (For a challenge, using Boolean operators can reduce this function to one line.)

`process_lyrics(lyrics, stopwords) -> set`
This function receives a string and a set of stop words as parameters. The string contains the lyrics. The function splits the lyrics by space. Each word is then made lower case and stripped of whitespace and then punctuation. After that it validates each word by using the validate_word function. If the word is valid, it adds that word to a set which will be returned after all words are processed.
Note: Do not forget to convert the words to lowercase and strip punctuation from the end of words (if there is something like a hyphen in the middle of a string, we will not strip it).
Hint: string.punctuation is useful.

`update_dictionary(data_dict, singer, song_name, song_words_set) -> None`
This function receives a data dictionary, singer's name, song's name and a set of words as parameters. The data_dict is a dictionary of singers (the key), and each value is a dictionary of all the signer's songs (song_dict). The song_dict is a dictionary of song_name: song_words_set key-value pairs. This function inserts a song_name: song_words_set key-value pair to the song_dict dictionary.

The following is how we want the data_dict to be formatted:
`{"singer1":{"song1": set_of_words, "song2": set_of_words,…}, "singer2":{…},…}`

The challenge for this function is that the first time this function is called the data_dict will initially be empty. If it isn't totally empty, it might be called for the first time for a singer, so a singer needs to be added. Is it a problem, if the song is already in the song_dict of a singer?

`read_data(fp, stopwords)->dict{str:{str:set, str:set,…},str:{…},…}`
This function has two parameters, the file pointer for a csv file and a set of stop words. It reads in the data collecting 3 things from each row: singer name as a string, song name as a string and lyrics as a string. You should iterate through the file row by row and for every row, you should read singer name, song name and the entire lyrics of that song which consists of many lines. You should convert the lyrics to lowercase. Then by using process_lyrics function you should process the lyrics to create a set of words. After that, you can update the dictionary by using update_dictionary function and passing the values you read. Finally, it should return the dictionary.

Note: Before starting the implementation, take a look at the csv file. Note that lyrics are stored in a specific format (lyrics are stored inside quotation marks). To make the reading process easy, you can use "csv.reader(fp)" to read the csv file row by row. Otherwise, reading the file will not be easy.

Example:

```
reader = csv.reader(fp)
for row in reader: # row is a list, i.e. you do not need split()
      column_0 = row[0]
```

```
            column_1 = row[1]
```
Note: to skip a line of the file, e.g. to skip headers, use `next(reader)`

`calculate_average_word_count(data_dict) ->dict{str:float,…}`
This function receives `data_dict` (which is created by the `read_data` function) and returns another dictionary which contains average word counts of singers.
We define average word count for a singer as the total number of words used by that singer, divided by the number of songs of the singer.

You should do this calculation for every singer and store the results in a dictionary and return it.
Here is the format of the dictionary:
```
dict{"singer1": average, "singer2": average,…}
```

`find_singers_vocab(data_dict) -> dict{str:set,…}`
This function receives data_dict (which is created by read_data function) and returns another dictionary which contains set of distinct words used by every singer. To create a set of vocabulary for a singer, you should find the union of all the words that are used by that singer. Finally, the function returns a dictionary in this format:
```
dict{"singer1": set1, "singer2": set2,…}
```

`display_singers(combined_list) -> None`
This function receives a list which is created in the main function and it includes a tuple for every singer. Each tuple should have the following data:
`(singer name, average word count, number of songs, vocabulary size)`
where vocabulary size is the number of distinct words used in the singer's songs.
This function sorts the list by average word count in descending order and if two tuples have the same average, it sorts them by the vocabulary size, again in descending order. Hint: itemgetter is useful for sorting by multiple items.
Finally, it prints the top ten tuples after sorting the list in the given format.

`search_songs(data_dict, words) -> list`
This function receives data_dict and a set of words which includes the words of the given query. It creates a list of tuples (singer name, song name) every time it finds a match. If a song includes every word in the given word set, you should include that song and the singer of that song in the output list. Before you return the list, you should sort it by singer name in alphabetical order and if two tuples have the same singer, you should sort them by song name (again, alphabetical order).
Hint: subset is useful

`main()`
In the main function, stop words and song data will be read by using the appropriate function calls. After that average word count and vocabulary will be calculated for every singer. To be able to display the results, the output of these functions should be combined in the following format: (singer name, number of songs, average word count, vocabulary size). Since there will be a tuple for every singer, you should create a list to store these tuples. The list could be used to display the results. Also, to prepare the inputs for plotting, you can use this list. After that part, you need to prompt to get a set of words to search through lyrics. After searching, you should print the top 5 results. Your program shouldn't crash if it returns fewer results. You can check the test cases for more information.

Test Cases
Test 1: No search, no plotting

```
Enter a filename for the stopwords: stopwords.txt
Enter a filename for the song data: songdata_small.csv


                  Singers by Average Word Count (TOP - 10)
Singer                Average Word Count     Vocabulary Size     Number of Songs
-------------------------------------------------------------------------------
Eminem                             172.19                4806                  70
Kanye West                         122.97                4565                 106
Nicki Minaj                        106.56                3858                  88
Justin Timberlake                   79.57                1845                  60
Bob Dylan                           74.29                4482                 188
Ed Sheeran                          71.96                1649                  53
Taylor Swift                        69.21                1791                  81
Rihanna                             66.37                2686                 143
Justin Bieber                       63.79                2537                 131
Katy Perry                          63.09                2258                  89
Do you want to plot (yes/no)?: no

Search Lyrics by Words



Input a set of words (space separated), press enter to exit:
```

Test 2: Searching songs (no error checking, no plotting)

```
Enter a filename for the stopwords: stopwords.txt
Enter a filename for the song data: songdata_small.csv

                  Singers by Average Word Count (TOP - 10)
Singer                Average Word Count     Vocabulary Size     Number of Songs
-------------------------------------------------------------------------------
Eminem                             172.19                4806                  70
Kanye West                         122.97                4565                 106
Nicki Minaj                        106.56                3858                  88
Justin Timberlake                   79.57                1845                  60
Bob Dylan                           74.29                4482                 188
Ed Sheeran                          71.96                1649                  53
Taylor Swift                        69.21                1791                  81
Rihanna                             66.37                2686                 143
Justin Bieber                       63.79                2537                 131
Katy Perry                          63.09                2258                  89
Do you want to plot (yes/no)?: no

Search Lyrics by Words

Input a set of words (space separated), press enter to exit: love hate like
There are 61 songs containing the given words!
Singer              Song
Adele               Someone Like You
Adele               Water Under The Bridge
Ariana Grande       Problem
Bob Dylan           I'm Not There
Britney Spears      If U Seek Amy
```

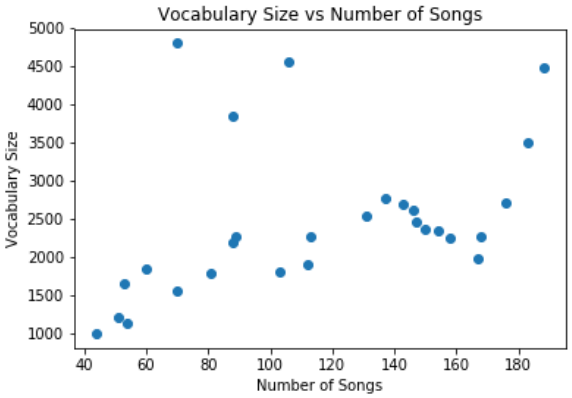
```
Input a set of words (space separated), press enter to exit: war peace
There are 6 songs containing the given words!
Singer               Song
Bob Dylan            Gates Of Eden
Christina Aguilera   Cease Fire
Ed Sheeran           Blind Faith
Katy Perry           Choose Your Battles
Michael Jackson      Earth Song

Input a set of words (space separated), press enter to exit:
```

Test 3: Complete test (with plotting)

```
Enter a filename for the stopwords: stopwords.txt
Enter a filename for the song data: songdata_small.csv

                    Singers by Average Word Count (TOP - 10)
Singer                 Average Word Count     Vocabulary Size     Number of Songs
-------------------------------------------------------------------------------
Eminem                             172.19                4806                  70
Kanye West                         122.97                4565                 106
Nicki Minaj                        106.56                3858                  88
Justin Timberlake                   79.57                1845                  60
Bob Dylan                           74.29                4482                 188
Ed Sheeran                          71.96                1649                  53
Taylor Swift                        69.21                1791                  81
Rihanna                             66.37                2686                 143
Justin Bieber                       63.79                2537                 131
Katy Perry                          63.09                2258                  89
Do you want to plot (yes/no)?: yes
```




```
Search Lyrics by Words

Input a set of words (space separated), press enter to exit: I love programming
Error in words!
1-) Words should not have any digit or punctuation
2-) Word list should not include any stop-word

Input a set of words (space separated), press enter to exit: love programming
There are 0 songs containing the given words!

Input a set of words (space separated), press enter to exit: LoVe
There are 1491 songs containing the given words!
Singer               Song
Adele                All I Ask
Adele                Best For Last
Adele                Can't Let Go
```

```
Adele                Chasing Pavements
Adele                First Love

Input a set of words (space separated), press enter to exit: roads man walk
There are 3 songs containing the given words!
Singer               Song
Bob Dylan            Blowin' In The Wind
Elvis Presley        Blowin' In The Wind
Johnny Cash          Highway Patrolman

Input a set of words (space separated), press enter to exit:
```

Function Tests: for each there is an input and an "instructor value" that the instructor's version of the function returned.

Function Test read_stopwords

```
Instructor values for read_stopwords(open("stopwords.txt")): {'other', "it's", 'them',
"he's", 'will', "won't", 'her', 'only', 'with', 'no', 'few', "they're", 'was', 'so', 'at',
'off', 'just', 'on', "they've", 'ourselves', 'both', "he'll", "that's", 'this', 'where',
'by', 'for', "weren't", "here's", "shouldn't", "he'd", "how's", 'theirs', "isn't", 'does',
"wouldn't", 'each', 'then', 'well', 'am', 'every', 'an', 'good', "haven't", 'our', 'yours',
'has', "who's", "let's", 'to', 'it', 'do', 'such', "ain't", "i'd", 'i', 'be', 'below',
'any', 'ours', "i've", "there's", 'would', 'how', 'are', 'against', 'their', 'again',
'until', 'themselves', 'there', 'than', "she'll", "i'm", "wasn't", 'further', 'they',
'those', 'before', 'hers', 'never', 'because', "hasn't", 'under', 'always', 'his',
'nothing', "mustn't", "shan't", "we're", 'is', 'which', 'he', 'should', 'into', 'could',
'these', 'everyone', 'himself', "couldn't", "they'd", 'when', 'who', "she'd", 'him',
'your', 'my', 'through', 'ought', 'its', 'over', "don't", 'ever', "aren't", 'more', 'once',
"can't", 'too', "doesn't", 'having', 'in', 'some', 'still', 'around', 'about', 'myself',
'aint', 'between', 'yourself', 'have', "she's", 'she', "you've", 'up', "they'll", 'out',
'us', "hadn't", "you're", 'from', 'nor', "where's", 'of', 'same', 'me', 'during', 'if',
'that', 'being', 'above', 'doing', 'and', 'herself', 'did', 'own', 'had', "we'd", 'now',
'already', 'what', 'or', 'as', "when's", "we've", 'not', 'we', 'while', 'most', "you'll",
'down', "what's", 'you', 'very', 'here', 'hey', 'a', 'oh', 'were', 'anything', 'cannot',
'the', 'all', "i'll", 'can', "we'll", "why's", 'been', 'why', 'but', "didn't", 'itself',
'whom', 'yourselves', "you'd", 'after'}
```

Function Test validate_word

```
After storing the stop words in "stopwords"

for w in ["you", "love", "peace!", "face2face"]:
    print(w, ":" , validate_word(w, stopwords))

Instructor values:
you : False
love : True
peace! : False
face2face : False
```

Function Test process_lyrics

```
After storing the stop words in "stopwords"

Testing string:
lyrics = """Yes, how many times must a man look up
Before he can really see the sky?
Yes, how many ears must one man have
Before he can hear people cry?
Yes, how many deaths will it take till he knows
That too many people have died?
The answer my friend is blowin' in the wind
```

```
The answer is blowin' in the wind."""

Instructor value for process_lyrics(lyrics, stopwords): {'look', 'times', 'hear', 'till',
'The', 'wind', 'must', 'really', 'many', 'see', 'one', 'deaths', 'man', 'take', 'answer',
'That', 'people', 'ears', 'knows', 'friend', 'Before'}
```

Function Test read_data

```
After storing the stop words in "stopwords"

Instructor value for read_data(open("songdata_test.csv"), stopwords):
{'Adele': {"Can't Let Go": {'let', 'note', 'hope', 'went', 'slow', 'coat', 'faked',
'throat', 'tell', 'yet', 'write', 'thinking', 'round', 'everything', 'lied', 'know',
'oooh', 'thought', 'dark', 'kill', 'lump', 'baby', 'seam', 'told', 'said', 'find',
'sometimes', 'time', 'even', 'platter', 'wanted', 'go', 'hid', 'like', 'hard', 'truth',
'arms', 'save', 'die', 'gave', 'loved', 'heaven', 'feel', 'life', 'much'}, 'All I Ask':
{'let', 'ends', 'get', 'cruel', 'one', 'run', 'play', 'ask', 'say', 'friend', 'use',
'need', 'tell', 'vicious', 'scared', 'honesty', 'next', 'last', 'love', 'take', 'know',
'tomorrow', 'memory', 'hand', 'lesson', 'wanna', 'forgiveness', 'leave', 'knows',
'matters', 'eyes', 'speak', 'since', 'said', 'heart', 'night', 'give', 'lovers', 'wrong',
'way', 'like', 'cause', 'left', 'remember', 'asking', 'door', 'word', 'coming', 'sure',
'pretend', 'hold'}}, 'Bob Dylan': {'4Th Time Around': {'shoe', 'forgotten', 'get', 'went',
'straightened', 'must', 'crutch', 'dirt', 'screamed', 'got', 'felt', 'ask', 'buttoned',
'jamaican', 'rum', 'pockets', 'look', 'better', 'till', 'wasted', 'waited', 'tried',
'last', 'finding', 'else', 'leaned', 'thought', 'drum', 'tapped', 'asked', 'gallantly',
'picture', 'breaking', 'cried', 'forced', 'something', 'piece', 'hands', 'everybody',
'words', 'filled', 'covered', 'worked', 'give', 'waste', 'wheelchair', 'face', 'brought',
'took', 'sense', 'back', 'spit', 'make', 'handed', 'go', 'fell', 'threw', 'stood', 'loved',
'leave'}, 'A Satisfied Mind': {'little', 'suddenly', 'lost', 'happened', 'ones', 'get',
'comes', 'far', 'one', 'many', 'wading', 'game', 'run', 'doubt', 'old', 'say', 'richer',
'heard', 'dreamed', 'everything', 'know', 'friends', 'world', 'satisfied', 'fame', 'money',
'times', 'dime', 'find', 'man', 'lifes', 'time', 'fortune', 'ten', 'way', 'hard',
'someone', 'things', 'start', 'loved', 'certain', 'life', 'leave'}}}
```

Function Test calculate_average_word_count

```
data_dict: {'Adele': {'All I Ask': {'use', 'honesty', 'scared', 'hand', 'tell', 'door',
'pretend', 'last', 'sure', 'lovers', 'leave', 'forgiveness', 'say', 'ask', 'night',
'vicious', 'ends', 'love', 'coming', 'left', 'knows', 'asking', 'lesson', 'memory',
'wanna', 'heart', 'eyes', 'let', 'take', 'matters', 'give', 'way', 'play', 'cruel',
'friend', 'run', 'since', 'get', 'cause', 'hold', 'one', 'said', 'next', 'wrong', 'like',
'remember', 'tomorrow', 'need', 'speak', 'know', 'word'}, "Can't Let Go": {'lied', 'write',
'lump', 'truth', 'even', 'baby', 'tell', 'gave', 'die', 'save', 'dark', 'heaven',
'thinking', 'round', 'platter', 'go', 'went', 'feel', 'everything', 'hid', 'seam', 'loved',
'let', 'sometimes', 'throat', 'hope', 'yet', 'thought', 'wanted', 'oooh', 'time', 'kill',
'find', 'note', 'told', 'coat', 'slow', 'said', 'much', 'faked', 'life', 'like', 'hard',
'arms', 'know'}}, 'Bob Dylan': {'4Th Time Around': {'threw', 'else', 'filled', 'dirt',
'forgotten', 'waste', 'wasted', 'last', 'must', 'leave', 'words', 'worked', 'breaking',
'handed', 'ask', 'back', 'pockets', 'gallantly', 'go', 'went', 'picture', 'crutch',
'jamaican', 'loved', 'stood', 'better', 'screamed', 'till', 'tapped', 'waited', 'give',
'got', 'straightened', 'sense', 'brought', 'cried', 'covered', 'thought', 'forced', 'took',
'get', 'rum', 'finding', 'something', 'spit', 'make', 'buttoned', 'tried', 'look', 'face',
'wheelchair', 'hands', 'drum', 'felt', 'shoe', 'fell', 'piece', 'everybody', 'leaned',
'asked'}, 'A Satisfied Mind': {'fame', 'ones', 'money', 'start', 'old', 'certain',
'dreamed', 'leave', 'say', 'game', 'things', 'someone', 'suddenly', 'man', 'world',
'lifes', 'comes', 'everything', 'dime', 'loved', 'richer', 'ten', 'little', 'way', 'run',
'lost', 'fortune', 'get', 'time', 'one', 'far', 'times', 'satisfied', 'find', 'wading',
'life', 'doubt', 'hard', 'friends', 'heard', 'happened', 'know', 'many'}}}

Instructor value for calculate_average_word_count(data_dict):
{'Adele': 48.0, 'Bob Dylan': 51.5}
```

Function Test find_singers_vocab

```
data_dict: Same as the previous one

Instructor value for find_singers_vocab(data_dict):
{'Adele': {'let', 'note', 'ends', 'went', 'slow', 'coat', 'faked', 'run', 'friend', 'tell',
'say', 'write', 'thinking', 'scared', 'next', 'last', 'lied', 'take', 'forgiveness',
'hand', 'dark', 'matters', 'seam', 'eyes', 'since', 'told', 'night', 'lovers', 'time',
'wrong', 'platter', 'way', 'hid', 'go', 'truth', 'cause', 'door', 'gave', 'word', 'coming',
'sure', 'pretend', 'feel', 'even', 'leave', 'hold', 'hope', 'get', 'cruel', 'one',
'throat', 'play', 'ask', 'use', 'need', 'yet', 'vicious', 'round', 'honesty', 'everything',
'love', 'know', 'oooh', 'thought', 'tomorrow', 'memory', 'wanna', 'lesson', 'kill',
'knows', 'lump', 'baby', 'speak', 'said', 'give', 'heart', 'find', 'sometimes', 'wanted',
'like', 'hard', 'arms', 'asking', 'remember', 'left', 'save', 'die', 'loved', 'heaven',
'life', 'much'}, 'Bob Dylan': {'lost', 'ones', 'went', 'comes', 'straightened', 'screamed',
'run', 'felt', 'say', 'buttoned', 'richer', 'heard', 'look', 'better', 'dreamed', 'wasted',
'tried', 'last', 'drum', 'tapped', 'satisfied', 'fame', 'forced', 'something', 'piece',
'money', 'everybody', 'covered', 'worked', 'dime', 'face', 'brought', 'took', 'man',
'time', 'make', 'ten', 'way', 'go', 'someone', 'start', 'threw', 'stood', 'certain',
'leave', 'little', 'suddenly', 'shoe', 'forgotten', 'get', 'happened', 'far', 'crutch',
'must', 'one', 'dirt', 'many', 'got', 'doubt', 'ask', 'wading', 'game', 'old', 'jamaican',
'rum', 'pockets', 'till', 'waited', 'everything', 'leaned', 'finding', 'else', 'know',
'thought', 'asked', 'picture', 'gallantly', 'friends', 'world', 'breaking', 'cried',
'filled', 'words', 'hands', 'times', 'give', 'waste', 'wheelchair', 'find', 'lifes',
'spit', 'sense', 'back', 'fortune', 'handed', 'hard', 'things', 'fell', 'loved', 'life'}}
```

Function Test search_songs

```
data_dict: Same as the previous one

Instructor value for search_songs(data_dict, {'heaven'}): [('Adele', "Can't Let Go")]

Instructor value for search_songs(data_dict, {'time'}): [('Adele', "Can't Let Go"), ('Bob
Dylan', 'A Satisfied Mind')]

Instructor value for search_songs(data_dict, {'time', 'tell', 'let' [('Adele', "Can't Let
Go")]
```