Lab 4: Essay Analyzer

English as a class subject is known to be very subjective, but the quality of an essay can be quantified and we can still analyze certain elements of an essay. Many websites out there allow you to upload your essay and give you a general analysis of it that may help you improve your essay or see where it stands (An example of this is: www.expresso-app.org). In this lab,we will be making a simple analyze that will give a general quantitative analysis of a provided essay.

In this lab, we will provide the following tools for an essay:

- Basic compositional stats
- Useful analytics
- Functional tools
- Add advanced analysis and tooling

Outcomes:

This lab is not necessarily more difficult than previous ones, but there is less guidance and a few advanced concepts. We will use various data structures and algorithms to efficiently solve problems and provide functionality.

Understand the skeleton code

You are provided 2 files:

1. **Doc-Tools.py:** skeleton code with all the expected method signatures and docstrings
    - I've provided a couple of testing methods to help you test your code.
    - Your program will be initalized with: tools = DocTools(<document>)
    - All other methods will be run from this instance
2. **doc1.txt**: An example input file
3. **dictionary.txt**: A text file with every word

Reminder: You may not import any modules not imported in the skeleton code.
However, *make sure to install matplotlib if you haven't already by hovering over the the text 'import matplotlib' and:*

- (if you have a Mac) holding "Option" + "Enter" at the same time
- (if you have a PC) holding "Alt" + "Enter" at the same time
- or try clicking "More Actions" and then selecting "import package"

As always, you may not import any modules other than the ones I've provided.

Suggested Helper Methods

I will not be providing helper methods this time, but this means you can create any helper method you want that may help you write the graded methods. I **highly** suggest the following:

- Process Document Helper Methods (from the provided list_of_paragraphs)
    - Generate a list of words (without punctuation)
    - Generate a list of clean words (all lowercase)
    - Generate a list of sentences (without punctuation)
- Make Regex (from a list)
    - To process sentences, it may be helpful to use re.split() which can split by a regex expression which allows you to split by multiple characters at once
    - A regex expression for a list of terminators would be '[.?!]

Category 1: Provided Methods (10 points)

You will get 10 free points for this section as this will probably take the longest if you can create the optional and suggested helper methods, but will not be tested by the autograder.

- init

  - Will take the name of the document, defines the punctuation and sentence terminating punctuation. Will call load document to read and create the provided list of paragraphs
  - Here, you should add other stored variables from processing the document (helper methods) so you can reuse common data

- load_document
  - Loads in a document and creates a list of paragraph strings by splitting by paragraph breaks
  - This method has been implemented for you. Please make sure you understand exactly what is happening.

Category 2: Basic Stats (30 points)

- word_count()
  - Returns the total number of words in the document (separated by spaces, paragraph breaks, punctuation, etc)
  - Expected count: 1268
- character_count()
  - Returns the total number of characters in the document, including spaces and punctuation, but not new line characters
  - Expected count: 7749
- sentence_count()
  - Returns the total number of sentences
  - Sentences are separated by the sentence terminating characters and paragraph breaks
  - Expected count: 47

Category 3: Essay Analysis (30 points)

- longest_word()
  - Returns the length of the longest word and a list of all words of the same length
  - Expected output: (['self-sufficient'], 15)
- average_word_length()
  - Returns the average length of the words used in the document, rounded to 2 decimal places
  - Expected output: 5.04
- average_sentence_length()
  - Returns the average length of sentences in words rounded to the nearest 2 decimal places

  - Expected output: 26.98

- longest_sentence()
  - Returns a tuple with the length of the longest sentence and a list of sentences of that same length
  - Expected output: (['Those who hold different conceptions ... '], 48)
- shortest_sentence()
  - Return a tuple with the length of the shortest sentence and a list of sentences of that same length
  - Expected output: (['No doubt they are expressed too strongly'], 7)
- most_frequent_words(n)
  - Returns the n most common words in the document, ignoring the most common words in the English language (common_words) to show distinctive words
  - Break any ties alphabetically
  - Expected output if n=5: ['justice', 'social', 'principles', 'these', 'their']
- num_distinct_words()
  - Returns the total number of unique words
  - Expected count: 448

Category 4: Advanced Analysis (15 points + 10 points of EC)

- words_by_prefix(prefix)
  - Return a list of all words that start with the parameter prefix, sorted alphabetically
  - Expected output: ['reason', 'recognize', 'reformed', 'regularly', 'regulated', ...]
- character_fingerprint() [Extra Credit]
  - Generate the alphabet fingerprint of the document, which is the frequency of each of the letters a-z as a dictionary
  - After generating the fingerprint, use matplotlib to create a bar chart where the x-axis is the letters in order from a to z and the y-axis is the frequency of the letters as a percentage. **Comment out the plotting code before submitting (including the import).**
  - Your function should only return the dictionary
  - Include the generated plot in your README
  - Expected dictionary output: 'j': 45, 'u': 183, 's': 506, 't': 643, ...
    - The order of the dictionary should not matter but the given letters should have the given values to them
- auto_complete(input_string) [Extra Credit]

  - We will generate a simple auto-complete function from the document. This will work similarity to PyCharm's autocomplete where we will suggest a few options that frequently occurred previously.
  - To do this, return a list of the top 3 most frequent words from the document that start with the parameter string.
  - This is a combination of the most used words and prefix methods.
  - When there are less than 3 words, only return the number of possible words in order (so if there are 2 words with the prefix, then just return [the most used word, less used word]
  - You should break ties alphabetically
  - Expected output: Auto Complete for "in": ['in', 'institutions', 'interests']

Category 5: Tools (15 points + 15 points of EC)

- find_word(word)
  - Find all instances of a word, ignoring capitalization and punctuation
  - This should be regular indices so 1-indexed (The 5th word in a sentence is 5 (instead of 4)
  - Expected output: [1, 57, 72, 131, 188, ...]
- replace_word(original, new) [Extra Credit]
  - For replace, we will not ignore capitalization, but will maintain the punctuation.
  - For this method, we will be altering the document representation (and must modify all the relevant lists)
  - Remember that words are buffered by spaces or punctuation in the paragraph/sentence representations so "injustice" should not be changed if you want to replace the word "justice"
  - You will be returning the number of changes made and the new altered text in a list (separated by paragraphs)
  - Expected output: 30 changes (also make sure you return the list with the updated words)
- spell_check() [Extra Credit]
  - Find all instances of misspelled words. We will be using a dictionary to find valid words
  - This should be regular indices so 1-indexed (5th word in a sentence is 5)
  - Capitalization and punctuation should be ignored
  - We will use a word list scraped from "dictionary.txt"
  - Some words may be missing and hyphenated words will all be word (and this is ok)
  - Return a tuple of the number of errors and a list of indices
  - Expected output: (18, [40, 164, 288, 454, 507, 542, 557, 599, 604, 668, 677, 732, 962, 965, 1068, 1126, 1136, 1145])
- write_document(file_name) [Extra Credit]

- Save the internal representation into a file
- Make sure to include new line characters between paragraphs
- Your new text file should have the updated doc1.txt with the replaced word

Submission Instructions

Submit 2 files:
Doc-Tools.py
README.pdf with any notes on the lab and the graph from character_fingerprint()

Submit both parts on Gradescope

At the top of the README and Doc-Tools.py, include the following information:

- Your name
- The name of any classmates you discussed the assignment with, or the words "no collaborators"
- A list of sources you used (textbooks, wikipedia, research papers, etc.) to solve the assignment, or the words "no sources"
- Whether or not you're using the extension

Grading Methodology

Each part of the lab will be graded by an automatic grading program. It will use the method signatures specified in this lab and will use multiple test cases. Each phase of the lab will be graded independently.

You can lose up to 10 points for not having proper comments and styling:

- Descriptive variable names
- Comments for what functions do
- Comments for complex parts of code
- Proper naming conventions for any variables, functions, and classes