Project 1: Word Anagrams

CS3753/CS5163: Data Science Summer 2023

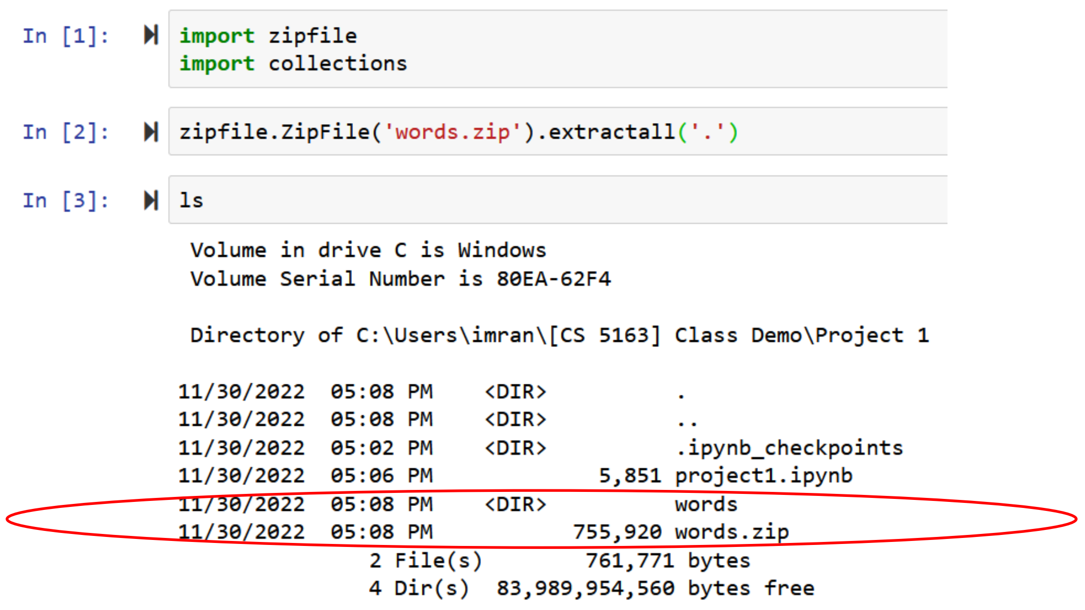
Instructor: Dr. Mohammad Imran Chowdhury

Total: 100 Points

Due: 06/13/2023 11:59 PM

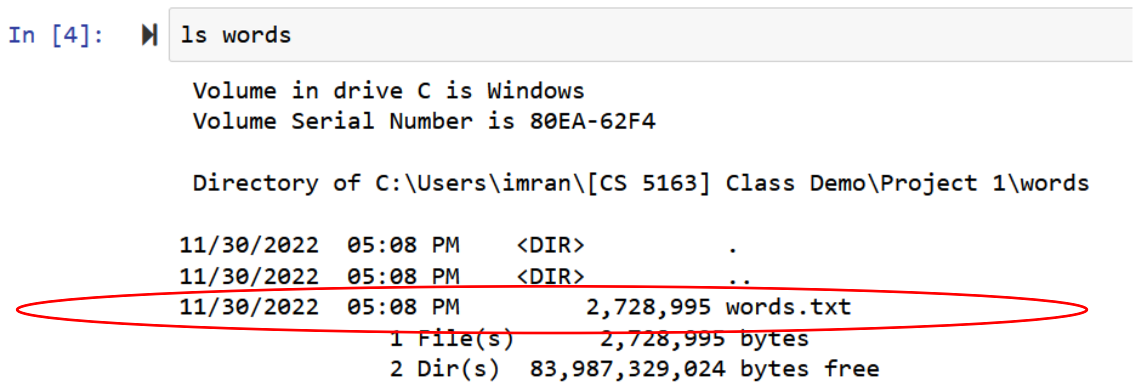
In this project, I invite you to do the following tasks using the Python data containers such as **List**, and **Dictionary** covered in class.

Task 1 (10 points): Load the Dataset Word Anagrams provided to you as the **``words.zip"** file into the Jupyter Notebook. Note that this notebook requires Python 3.6 or higher. After loading the dataset into the Jupyter Notebook, the output should look as follows: **(5 points)**



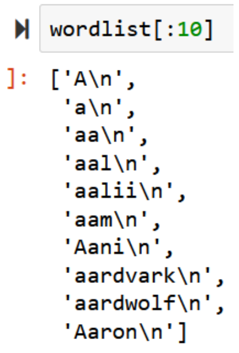
```
In [1]:  import zipfile
         import collections

In [2]:  zipfile.ZipFile('words.zip').extractall('.')

In [3]:  ls

 Volume in drive C is Windows
 Volume Serial Number is 80EA-62F4

 Directory of C:\Users\imran\[CS 5163] Class Demo\Project 1

11/30/2022  05:08 PM    <DIR>          .
11/30/2022  05:08 PM    <DIR>          ..
11/30/2022  05:02 PM    <DIR>          .ipynb_checkpoints
11/30/2022  05:06 PM             5,851 project1.ipynb
11/30/2022  05:08 PM    <DIR>          words
11/30/2022  05:08 PM           755,920 words.zip
               2 File(s)        761,771 bytes
               4 Dir(s)  83,989,954,560 bytes free
```

Type ``ls words" to list all files under the unzipped words folder. The output should be as follows: **(5 points)**



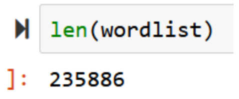
```
In [4]:  ls words

 Volume in drive C is Windows
 Volume Serial Number is 80EA-62F4

 Directory of C:\Users\imran\[CS 5163] Class Demo\Project 1\words

11/30/2022  05:08 PM    <DIR>          .
11/30/2022  05:08 PM    <DIR>          ..
11/30/2022  05:08 PM         2,728,995 words.txt
               1 File(s)      2,728,995 bytes
               2 Dir(s)  83,987,329,024 bytes free
```

Task 2 (20 points): Load/Read a dictionary of English words from the file 'words/words.txt' into a Python List as **wordlist**. The output should be as follows for the first ten (10) words and total number of words in the wordlist should match exactly with mine: **(10 points)**



```
wordlist[:10]
```

```
]: ['A\n',
    'a\n',
    'aa\n',
    'aal\n',
    'aalii\n',
    'aam\n',
    'Aani\n',
    'aardvark\n',
    'aardwolf\n',
    'Aaron\n']
```



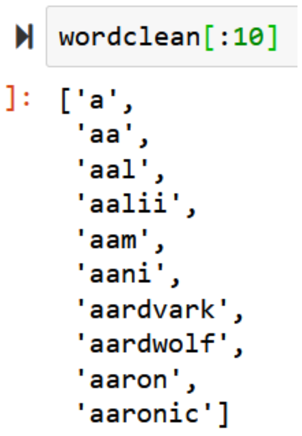
```
len(wordlist)
```

```
]: 235886
```

After that, you need to do some cleaning on the **wordlist**. These are as follows:

(a) Remove all the leading and trailing spaces including the `\n' character. This involves the use of the *strip()* method
(b) Make them i.e., words all lowercase
(c) Remove all duplicates
(d) Finally, sort them in lexicographic order means in alphabetical order, and save the result into a new List called **wordclean**.

To do the above steps, you can re-use the class coding demo file available on the Blackboard course website. After doing the cleaning on the wordlist, the output for the first ten (10) words should be as follows: **(10 points)**

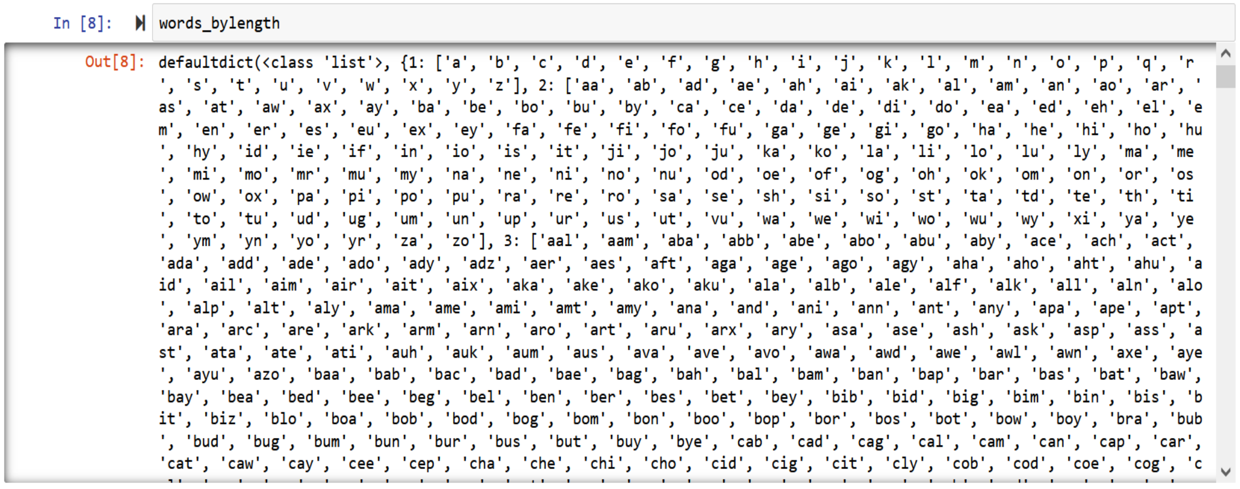


```
wordclean[:10]
```

```
]: ['a',
    'aa',
    'aal',
    'aalii',
    'aam',
    'aani',
    'aardvark',
    'aardwolf',
    'aaron',
    'aaronic']
```

Task 3 (20 points): From the generated **List wordclean**, separate words into classes of words with the same length. This involves creating of a **Dictionary** called **words_bylength** from the

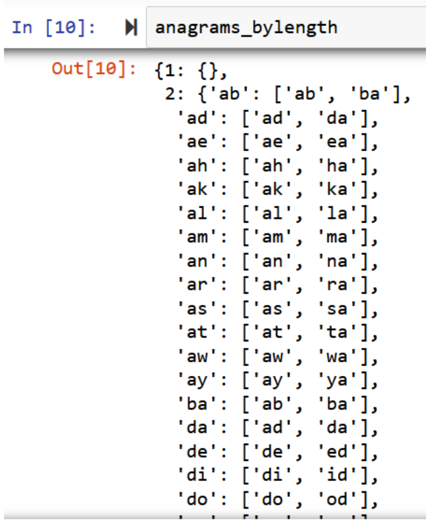
wordclean List where key is the word length and the value is the set of words having the same word length. By doing so, the output should look as follows: **(20 points)**



```
In [8]:  words_bylength

Out[8]:  defaultdict(<class 'list'>, {1: ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r
         ', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'], 2: ['aa', 'ab', 'ad', 'ae', 'ah', 'ai', 'ak', 'al', 'am', 'an', 'ao', 'ar', '
         as', 'at', 'aw', 'ax', 'ay', 'ba', 'be', 'bo', 'bu', 'by', 'ca', 'ce', 'da', 'de', 'di', 'do', 'ea', 'ed', 'eh', 'el', 'e
         m', 'en', 'er', 'es', 'eu', 'ex', 'ey', 'fa', 'fe', 'fi', 'fo', 'fu', 'ga', 'ge', 'gi', 'go', 'ha', 'he', 'hi', 'ho', 'hu
         ', 'hy', 'id', 'ie', 'if', 'in', 'io', 'is', 'it', 'ji', 'jo', 'ju', 'ka', 'ko', 'la', 'li', 'lo', 'lu', 'ly', 'ma', 'me
         ', 'mi', 'mo', 'mr', 'mu', 'my', 'na', 'ne', 'ni', 'no', 'nu', 'od', 'oe', 'of', 'og', 'oh', 'ok', 'om', 'on', 'or', 'os
         ', 'ow', 'ox', 'pa', 'pi', 'po', 'pu', 'ra', 're', 'ro', 'sa', 'se', 'sh', 'si', 'so', 'st', 'ta', 'td', 'te', 'th', 'ti
         ', 'to', 'tu', 'ud', 'ug', 'um', 'un', 'up', 'ur', 'us', 'ut', 'vu', 'wa', 'we', 'wi', 'wo', 'wu', 'wy', 'xi', 'ya', 'ye
         ', 'ym', 'yn', 'yo', 'yr', 'za', 'zo'], 3: ['aal', 'aam', 'aba', 'abb', 'abe', 'abo', 'abu', 'aby', 'ace', 'ach', 'act',
         'ada', 'add', 'ade', 'ado', 'ady', 'adz', 'aer', 'aes', 'aft', 'aga', 'age', 'ago', 'agy', 'aha', 'aho', 'aht', 'ahu', 'a
         id', 'ail', 'aim', 'air', 'ait', 'aix', 'aka', 'ake', 'ako', 'aku', 'ala', 'alb', 'ale', 'alf', 'alk', 'all', 'aln', 'alo
         ', 'alp', 'alt', 'aly', 'ama', 'ame', 'ami', 'amt', 'amy', 'ana', 'and', 'ani', 'ann', 'ant', 'any', 'apa', 'ape', 'apt',
         'ara', 'arc', 'are', 'ark', 'arm', 'arn', 'aro', 'art', 'aru', 'arx', 'ary', 'asa', 'ase', 'ash', 'ask', 'asp', 'ass', 'a
         st', 'ata', 'ate', 'ati', 'auh', 'auk', 'aum', 'aus', 'ava', 'ave', 'avo', 'awa', 'awd', 'awe', 'awl', 'awn', 'axe', 'aye
         ', 'ayu', 'azo', 'baa', 'bab', 'bac', 'bad', 'bae', 'bag', 'bah', 'bal', 'bam', 'ban', 'bap', 'bar', 'bas', 'bat', 'baw',
         'bay', 'bea', 'bed', 'bee', 'beg', 'bel', 'ben', 'ber', 'bes', 'bet', 'bey', 'bib', 'bid', 'big', 'bim', 'bin', 'bis', 'b
         it', 'biz', 'blo', 'boa', 'bob', 'bod', 'bog', 'bom', 'bon', 'boo', 'bop', 'bor', 'bos', 'bot', 'bow', 'boy', 'bra', 'bub
         ', 'bud', 'bug', 'bum', 'bun', 'bur', 'bus', 'but', 'buy', 'bye', 'cab', 'cad', 'cag', 'cal', 'cam', 'can', 'cap', 'car',
         'cat', 'caw', 'cay', 'cee', 'cep', 'cha', 'che', 'chi', 'cho', 'cid', 'cig', 'cit', 'cly', 'cob', 'cod', 'coe', 'cog', 'c
```

Task 4 (25 points): For each class of words of the same length, find all anagrams. This involves creating of another **Dictionary** called **anagrams_bylength** from the **words_bylength Dictionary** where key is the value of each word in the **words_bylength Dictionary** and the value is the all-possible anagrams for that corresponding key.

For example, from the above output **words_bylength[2] = [aa, ab, ad, ae,……..]** where say, **ab** will be the key of the **Dictionary anagrams_bylength** and the value will be the all-possible anagrams i.e., **[ab, ba]**. Therefore, the final output of the **Dictionary anagrams_bylength** for each class of words of the same length will be as follows: **(25 points)**

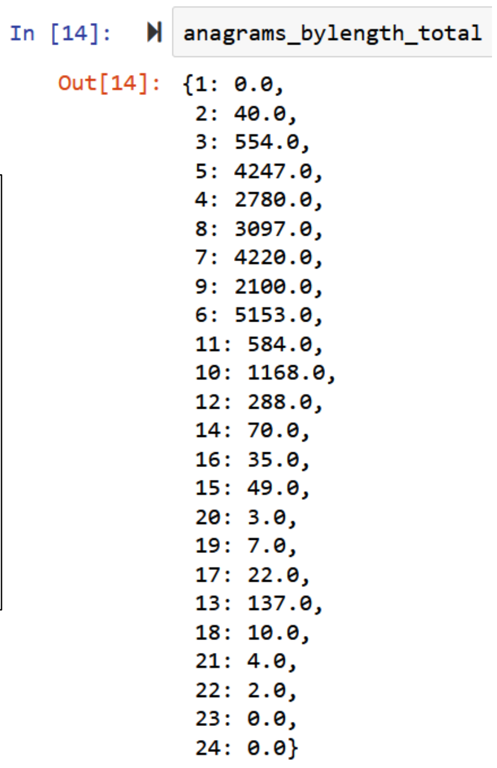


```
In [10]:  anagrams_bylength

Out[10]:  {1: {},
           2: {'ab': ['ab', 'ba'],
            'ad': ['ad', 'da'],
            'ae': ['ae', 'ea'],
            'ah': ['ah', 'ha'],
            'ak': ['ak', 'ka'],
            'al': ['al', 'la'],
            'am': ['am', 'ma'],
            'an': ['an', 'na'],
            'ar': ['ar', 'ra'],
            'as': ['as', 'sa'],
            'at': ['at', 'ta'],
            'aw': ['aw', 'wa'],
            'ay': ['ay', 'ya'],
            'ba': ['ab', 'ba'],
            'da': ['ad', 'da'],
            'de': ['de', 'ed'],
            'di': ['di', 'id'],
            'do': ['do', 'od'],
```

Note, to find all anagrams, you can re-use the **anagram_fast(myword)** method as shown in class. Also, to get the same output as mine, you've to consider the following condition: **len(anagram_fast(myword)) > 1** as well.

Tasks 5 (25 points): Now, count the total number of anagrams in each class from the **Dictionary words_bylength**. Again, this involves creating of another **Dictionary** called **anagrams_bylength_total** from the **words_bylength Dictionary** where the key is the length of each class of words where words having the same length and the value is the total number of anagrams in each class of words.

Your final program output should be exactly as follows: **(25 points)**



```
In [14]:  anagrams_bylength_total

Out[14]: {1: 0.0,
          2: 40.0,
          3: 554.0,
          5: 4247.0,
          4: 2780.0,
          8: 3097.0,
          7: 4220.0,
          9: 2100.0,
          6: 5153.0,
          11: 584.0,
          10: 1168.0,
          12: 288.0,
          14: 70.0,
          16: 35.0,
          15: 49.0,
          20: 3.0,
          19: 7.0,
          17: 22.0,
          13: 137.0,
          18: 10.0,
          21: 4.0,
          22: 2.0,
          23: 0.0,
          24: 0.0}
```

Note: Here, to get the right count of the total number of anagrams, you've to remove duplicates anagram counts for each key (same length of words), including the self-replica count. This is also discussed in class.

The submission grading rubric is as follows (points out of 100 total):

Project element	Points
Task 1	10
Task 2	20
Task 3	20
Task 4	25
Task 5	25

Submission Instructions: Create a compressed file (.zip or .tar.gz files are accepted) with your all source files such as .ipynb files and data files. Generally speaking to complete Task1 through Task5, you just need one .ipynb file. But it's better to submit everything as a compressed file. Submit the compressed file to Blackboard.

Late submission policy: As described in the syllabus, any late submission will the penalized with 10% off after each 24 hours late. For example, an assignment worth 100 points turned in 2 days late will receive a 20 point penalty. Assignments turned in 5 or more days after the due date will receive a grade of 0.