ADVANCED DATA MINING

ASSIGNMENT REPORT

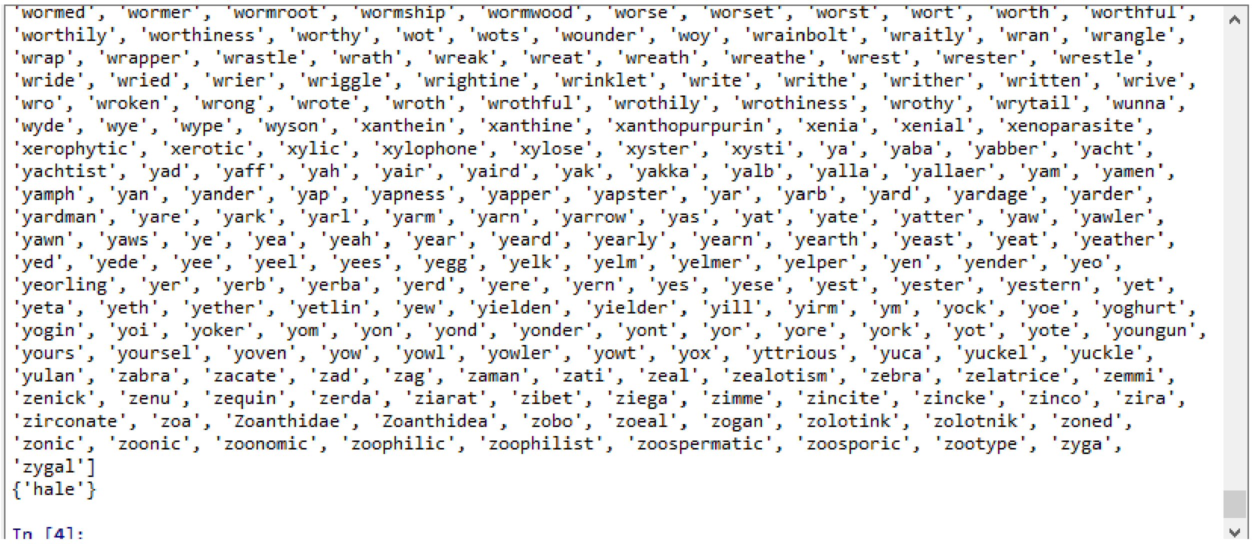
SUMANTH POBALA
UMID 71652304
DATA SCIENCE SEMESTER 1

DATA EXPLORATION ASSIGNMENT

1. Finding Anagrams using python

1.1. Firstly we will import the file to fname and define a function to initialize and all the words and define other functions for creating dictionary and finding anagrams and get all anagrams from dictionary and we will store all anagrams in an attribute called allanas and create a text file and store all the anagrams in it.

1.2. We can use my_key to find the anagram of a specific word where word will be searched in anagram dictionary.

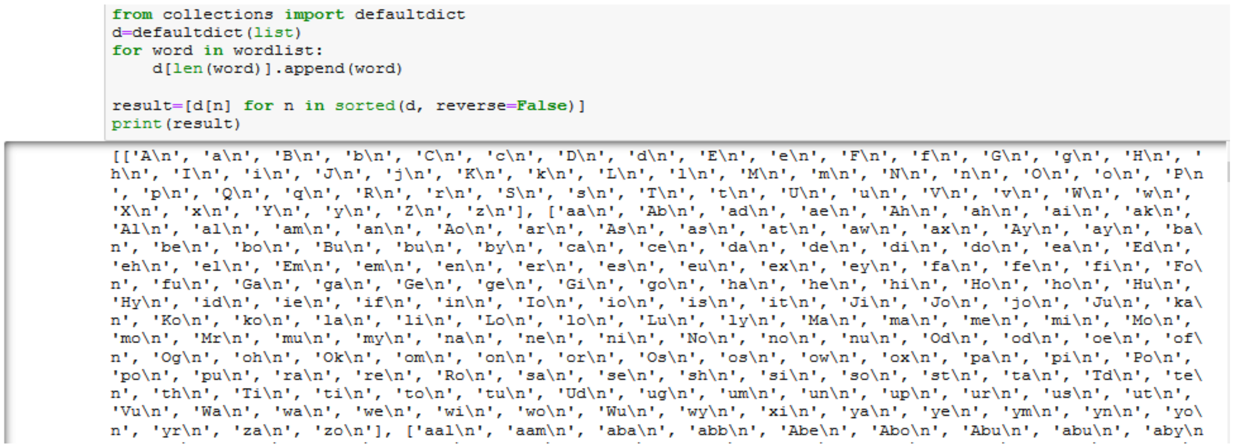


```
'wormed', 'wormer', 'wormroot', 'wormship', 'wormwood', 'worse', 'worset', 'worst', 'wort', 'worth', 'worthful',
'worthily', 'worthiness', 'worthy', 'wot', 'wots', 'wounder', 'woy', 'wrainbolt', 'wraitly', 'wran', 'wrangle',
'wrap', 'wrapper', 'wrastle', 'wrath', 'wreak', 'wreat', 'wreath', 'wreathe', 'wrest', 'wrester', 'wrestle',
'wride', 'wried', 'wrier', 'wriggle', 'wrightine', 'wrinklet', 'write', 'writhe', 'writher', 'written', 'wrive',
'wro', 'wroken', 'wrong', 'wrote', 'wroth', 'wrothful', 'wrothily', 'wrothiness', 'wrothy', 'wrytail', 'wunna',
'wyde', 'wye', 'wype', 'wyson', 'xanthein', 'xanthine', 'xanthopurpurin', 'xenia', 'xenial', 'xenoparasite',
'xerophytic', 'xerotic', 'xylic', 'xylophone', 'xylose', 'xyster', 'xysti', 'ya', 'yaba', 'yabber', 'yacht',
'yachtist', 'yad', 'yaff', 'yah', 'yair', 'yaird', 'yak', 'yakka', 'yalb', 'yalla', 'yallaer', 'yam', 'yamen',
'yamph', 'yan', 'yander', 'yap', 'yapness', 'yapper', 'yapster', 'yar', 'yarb', 'yard', 'yardage', 'yarder',
'yardman', 'yare', 'yark', 'yarl', 'yarm', 'yarn', 'yarrow', 'yas', 'yat', 'yate', 'yatter', 'yaw', 'yawler',
'yawn', 'yaws', 'ye', 'yea', 'yeah', 'year', 'yeard', 'yearly', 'yearn', 'yearth', 'yeast', 'yeat', 'yeather',
'yed', 'yede', 'yee', 'yeel', 'yees', 'yegg', 'yelk', 'yelm', 'yelmer', 'yelper', 'yen', 'yender', 'yeo',
'yeorling', 'yer', 'yerb', 'yerba', 'yerd', 'yere', 'yern', 'yes', 'yese', 'yest', 'yester', 'yestern', 'yet',
'yeta', 'yeth', 'yether', 'yetlin', 'yew', 'yielden', 'yielder', 'yill', 'yirm', 'ym', 'yock', 'yoe', 'yoghurt',
'yogin', 'yoi', 'yoker', 'yom', 'yon', 'yond', 'yonder', 'yont', 'yor', 'yore', 'york', 'yot', 'yote', 'youngun',
'yours', 'yoursel', 'yoven', 'yow', 'yowl', 'yowler', 'yowt', 'yox', 'yttrious', 'yuca', 'yuckel', 'yuckle',
'yulan', 'zabra', 'zacate', 'zad', 'zag', 'zaman', 'zati', 'zeal', 'zealotism', 'zebra', 'zelatrice', 'zemmi',
'zenick', 'zenu', 'zequin', 'zerda', 'ziarat', 'zibet', 'ziega', 'zimme', 'zincite', 'zincke', 'zinco', 'zira',
'zirconate', 'zoa', 'Zoanthidae', 'Zoanthidea', 'zobo', 'zoeal', 'zogan', 'zolotink', 'zolotnik', 'zoned',
'zonic', 'zoonic', 'zoonomic', 'zoophilic', 'zoophilist', 'zoospermatic', 'zoosporic', 'zootype', 'zyga',
'zygal']
{'hale'}

In [4]:
```

Fig.1 Anagrams and anagram of heal

1.3. Sorting the words based on the length, for doing that we will import the defualtdict which is located in collections and store output into 'result'.



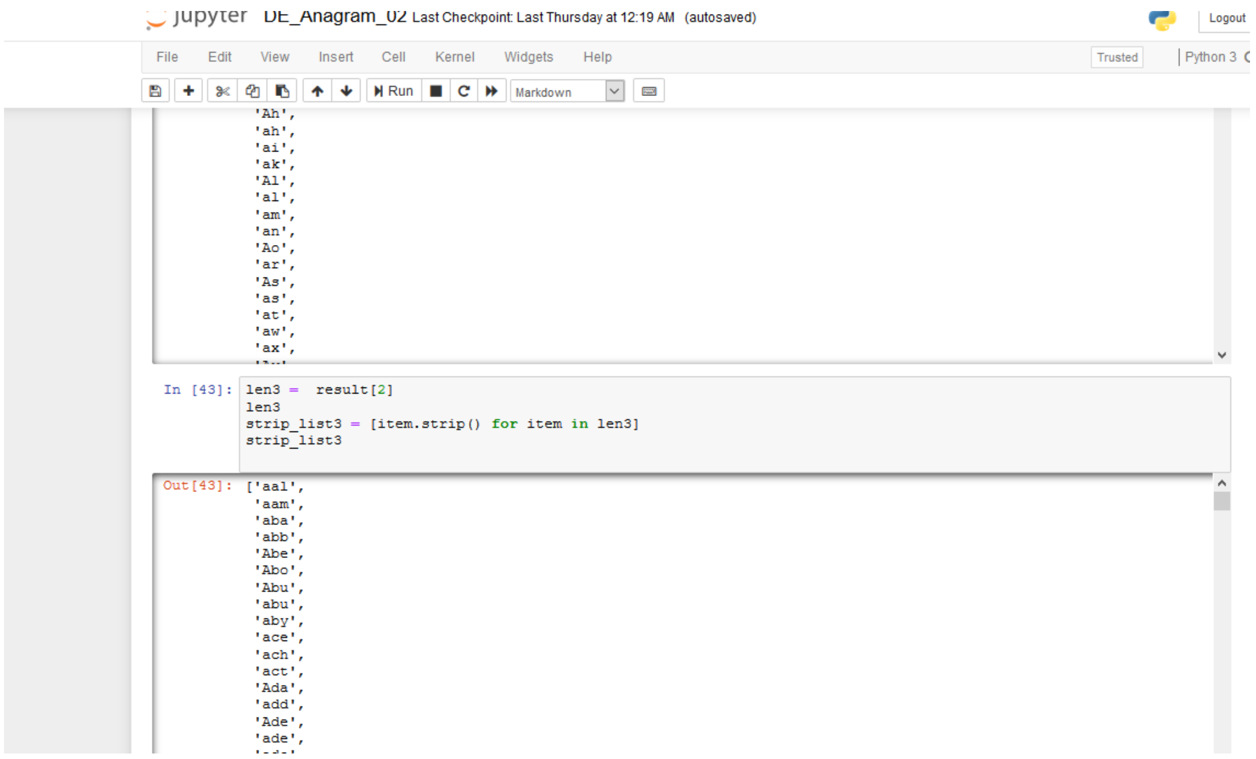
```
from collections import defaultdict
d=defaultdict(list)
for word in wordlist:
    d[len(word)].append(word)

result=[d[n] for n in sorted(d, reverse=False)]
print(result)
```

```
[['A\n', 'a\n', 'B\n', 'b\n', 'C\n', 'c\n', 'D\n', 'd\n', 'E\n', 'e\n', 'F\n', 'f\n', 'G\n', 'g\n', 'H\n', '
h\n', 'I\n', 'i\n', 'J\n', 'j\n', 'K\n', 'k\n', 'L\n', 'l\n', 'M\n', 'm\n', 'N\n', 'n\n', 'O\n', 'o\n', 'P\n
', 'p\n', 'Q\n', 'q\n', 'R\n', 'r\n', 'S\n', 's\n', 'T\n', 't\n', 'U\n', 'u\n', 'V\n', 'v\n', 'W\n', 'w\n',
'X\n', 'x\n', 'Y\n', 'y\n', 'Z\n', 'z\n'], ['aa\n', 'Ab\n', 'ad\n', 'ae\n', 'Ah\n', 'ah\n', 'ai\n', 'ak\n',
'Al\n', 'al\n', 'am\n', 'an\n', 'Ao\n', 'ar\n', 'As\n', 'as\n', 'at\n', 'aw\n', 'ax\n', 'Ay\n', 'ay\n', 'ba\
n', 'be\n', 'bo\n', 'Bu\n', 'bu\n', 'by\n', 'ca\n', 'ce\n', 'da\n', 'de\n', 'di\n', 'do\n', 'ea\n', 'Ed\n',
'eh\n', 'el\n', 'Em\n', 'em\n', 'en\n', 'er\n', 'es\n', 'eu\n', 'ex\n', 'ey\n', 'fa\n', 'fe\n', 'fi\n', 'Fo\
n', 'fu\n', 'Ga\n', 'ga\n', 'Ge\n', 'ge\n', 'Gi\n', 'go\n', 'ha\n', 'he\n', 'hi\n', 'Ho\n', 'ho\n', 'Hu\n',
'Hy\n', 'id\n', 'ie\n', 'if\n', 'in\n', 'Io\n', 'io\n', 'is\n', 'it\n', 'Ji\n', 'Jo\n', 'jo\n', 'Ju\n', 'ka\
n', 'Ko\n', 'ko\n', 'la\n', 'li\n', 'Lo\n', 'lo\n', 'Lu\n', 'ly\n', 'Ma\n', 'ma\n', 'me\n', 'mi\n', 'Mo\n',
'mo\n', 'Mr\n', 'mu\n', 'my\n', 'na\n', 'ne\n', 'ni\n', 'No\n', 'no\n', 'nu\n', 'Od\n', 'od\n', 'oe\n', 'of\
n', 'Og\n', 'oh\n', 'Ok\n', 'om\n', 'on\n', 'or\n', 'Os\n', 'os\n', 'ow\n', 'ox\n', 'pa\n', 'pi\n', 'Po\n',
'po\n', 'pu\n', 'ra\n', 're\n', 'Ro\n', 'sa\n', 'se\n', 'sh\n', 'si\n', 'so\n', 'st\n', 'ta\n', 'Td\n', 'te\
n', 'th\n', 'Ti\n', 'ti\n', 'to\n', 'tu\n', 'Ud\n', 'ug\n', 'um\n', 'un\n', 'up\n', 'ur\n', 'us\n', 'ut\n',
'Vu\n', 'Wa\n', 'wa\n', 'we\n', 'wi\n', 'wo\n', 'Wu\n', 'wy\n', 'xi\n', 'ya\n', 'ye\n', 'ym\n', 'yn\n', 'yo\
n', 'yr\n', 'za\n', 'zo\n'], ['aal\n', 'aam\n', 'aba\n', 'abb\n', 'Abe\n', 'Abo\n', 'Abu\n', 'abu\n', 'aby\n
```

Fig 1.1. Sorting Anagrams

1.4. Creating the sublists based on the indexing and strip the data using the strip function

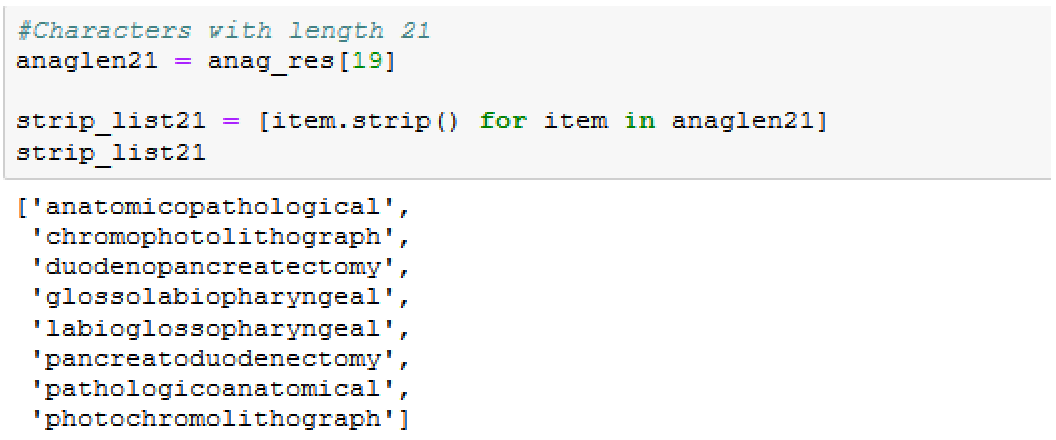

Jupyter DE_Anagram_02 Last Checkpoint: Last Thursday at 12:19 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown

```
           'Ah',
           'ah',
           'ai',
           'ak',
           'Al',
           'al',
           'am',
           'an',
           'Ao',
           'ar',
           'As',
           'as',
           'at',
           'aw',
           'ax',
```

```
In [43]: len3 =  result[2]
         len3
         strip_list3 = [item.strip() for item in len3]
         strip_list3
```

```
Out[43]: ['aal',
          'aam',
          'aba',
          'abb',
          'Abe',
          'Abo',
          'Abu',
          'abu',
          'aby',
          'ace',
          'ach',
          'act',
          'Ada',
          'add',
          'Ade',
          'ade',
```

Fig 1.2. Dividing into sublists (Before finding anagrams)



```
#Characters with length 21
anaglen21 = anag_res[19]

strip_list21 = [item.strip() for item in anaglen21]
strip_list21
```

```
['anatomicopathological',
 'chromophotolithograph',
 'duodenopancreatectomy',
 'glossolabiopharyngeal',
 'labioglossopharyngeal',
 'pancreatoduodenectomy',
 'pathologicoanatomical',
 'photochromolithograph']
```

Fig 1.3. Fig 1.2. Dividing into sublists (After finding anagrams)

2. Lengths of anagrams and sublists after finding anagrams

2.1. Length of words before finding anagrams : 235886 words

2.2. Total anagrams found : 27400

2.3. Length of anagrams of word length 2: 68

2.4. Length of anagrams of word length 3: 682

2.5. Length of anagrams of word length 4: 2341

2.6. Length of anagrams of word length 5: 3560

2.7. Length of anagrams of word length 6: 5137

2.8. Length of anagrams of word length 7: 4700

2.9. Length of anagrams of word length 8: 3983

2.10. Length of anagrams of word length 9: 3011

2.11. Length of anagrams of word length 10: 1837

2.12. Length of anagrams of word length 11: 930

2.13. Length of anagrams of word length 12: 519

2.14. Length of anagrams of word length 13: 243

2.15. Length of anagrams of word length 14: 140

2.16. Length of anagrams of word length 15: 86

2.17. Length of anagrams of word length 16: 70

2.18. Length of anagrams of word length 17: 42

2.19. Length of anagrams of word length 18: **20**

2.20. Length of anagrams of word length 19: **14**

2.21. Length of anagrams of word length 20: **6**

2.22. Length of anagrams of word length 21: **8**

2.23. Length of anagrams of word length 22: **4**