**ENSF 592**

**Assignment 3: Data Structure Selection**

**Asif Bux**

* 1. **Revisiting Assignment02 (0.5pt)**

There are actually few things I can think of I would differently after reading and doing the project Gutenberg exercises in the textbook , and as well as going over the solutions presented. There are two areas where I could have written more efficient code, first of all, I had to calculate the total of personal pronouns in the function called numOfPersonalPronouns():, instead of using “for elements of pronoun I text”, I could have used the dictionary to get the key and the value, and then summed up all the values to gather to find total. In this fashion, I would have also generated a record of which pronoun is common or infrequent. The second area, where I could have definitely improved my algorithm was in Avoids problem. I used counter() to keep track of total value of each alphabet but instead I could have simply set a dictionary with {alphabets: frequency}, I would then converted the key and value pair into tuple and then finally place all tuples in a list. Then I could have used sort() to find out the common alphabets and sliced for ones I needed while using the reverse().

* 1. **Analyze Feynman – again (1pt)**

To run the markov.py script I had to change the filename to the correct file as it is stored on my computer directory. I also had to update the time for skip header function to startwith (“This two-year”) as well as the end line to startwith (“world?”)

Screenshot of the output:

accurate it is, how it fits into everything else, and how it may be changed when we later concentrate on some particular point is interesting, and how it fits into everything else, and how it may be changed when we learn more. Let us now proceed with our outline, or general map, of our understanding of the laws as yet. Therefore, things must be learned only to be incorrect. Mass is found to be incorrect. Mass is found to increase with velocity, but appreciable increases require velocities near that of light. A true law is: if an object moves with

[Finished in 0.143s]

To run the analyze\_book1.py script I had to change to the filename to the correct file name. Since, my file did not have anyheader I modified the skip\_header to false and then the code does not check for header. I made the end to startwith (“world?”)

Screenshot of the output:

Total number of words: 1076

Number of different words: 385

The most common words are:

the 67

of 44

to 35

is 35

we 25

and 22

a 22

in 21

that 20

it 17

but 13

so 12

you 11

laws 11

be 11

are 11

first 10

this 9

all 9

physics 8

The words in the book that aren't in the word list are:

Here are some random words from the book

can one velocities so following euclidean that state increase without of the within the go is of first is have how extent—that possible developing and with are million ideas may circumstances approximation a makes first results the “feel” one complete to a and else that point but the course mass kind we of and get it an chapters approximate it itself first so but are said every physics first is are shall we fun such some notice found that to be change there that apparatus physics us so fact unlearned test learned very these a map reasons more mass kind

* 1. **Character Frequency- (1pt)**

Screenshot of the output:

All Characters in the words.txt: {'a': 68582, 'h': 20200, 'e': 106758, 'd': 34552, 'i': 77412, 'n': 60513, 'g': 27848, 's': 86547, 'l': 47011, 'r': 64965, 'v': 9186, 'k': 9370, 'w': 8535, 'o': 54542, 'f': 12714, 'b': 17798, 'c': 34287, 'u': 31161, 't': 57059, 'm': 24741, 'p': 25789, 'y': 13473, 'x': 2700, 'j': 1780, 'z': 3750, 'q': 1632}

Sum of all characters in the text: 902905

The ten most common characters: [('e', 106758), ('s', 86547), ('i', 77412), ('a', 68582), ('r', 64965), ('n', 60513), ('t', 57059), ('o', 54542), ('l', 47011), ('d', 34552)]

The ten least common characters: [('q', 1632), ('j', 1780), ('x', 2700), ('z', 3750), ('w', 8535), ('v', 9186), ('k', 9370), ('f', 12714), ('y', 13473), ('b', 17798)]

[Finished in 0.308s]

**2.0 Brevity Law (2.5pt)**

Code Description:

1. The program first processes and cleans the text file to get it ready for dictionary where gives out the keys as words and values as their frequency.
2. The word length is also calculated using lenOfWords() that returns a dictionary word with their length
3. Frequency list and word length is created to feed in the Spearman analysis function. The frequency list is sorted to maximum of 100 and highest order. The word length is just sliced to maximum of 100 without sorting to ensure its index analysis is correct.
4. Now we have all the necessary data to begin our Spearman analysis. The algorithm for this is same as from the Wikipedia as mentioned on the assignment handout.

There is clear evidence that there is some degree of correlation between the frequency and word length due to a negative spearman value and spearman coefficient from scipy being in range of the calculated value. However, I feel there were some limitation from this data analysis such we did not considered the effect of word length that are same with exact same rank and how those data point could have been normalized to accurately reflect the spearman coefficient. As well as, outliners and values that fall out of selected standard deviation for the analyses.

Screenshot of the output:

Rank of Frequency: [100, 99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59, 57, 58, 56, 55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 44, 45, 43, 42, 40, 41, 39, 38, 37, 36, 34, 35, 33, 32, 30, 31, 29, 28, 27, 26, 25, 22, 23, 24, 21, 20, 17, 18, 19, 15, 16, 13, 14, 10, 11, 12, 9, 7, 8, 6, 5, 1, 2, 3, 4]

Rank of Word Length: [26, 85, 76, 58, 11, 27, 28, 1, 94, 29, 48, 95, 59, 2, 49, 30, 3, 12, 31, 86, 87, 13, 50, 60, 61, 14, 51, 4, 52, 32, 62, 77, 5, 15, 16, 17, 18, 78, 19, 88, 33, 99, 89, 63, 96, 79, 80, 34, 81, 20, 53, 35, 54, 64, 65, 36, 21, 37, 22, 23, 38, 39, 40, 6, 93, 97, 82, 55, 83, 7, 90, 56, 8, 91, 24, 66, 57, 92, 72, 41, 67, 9, 98, 68, 42, 69, 43, 44, 25, 100, 73, 45, 10, 84, 74, 46, 70, 71, 47, 75]

Difference di: [74, 14, 22, 39, 85, 68, 66, 92, -2, 62, 42, -6, 29, 85, 37, 55, 81, 71, 51, -5, -7, 66, 28, 17, 15, 61, 23, 69, 20, 39, 8, -8, 63, 52, 50, 48, 46, -15, 43, -27, 27, -40, -32, -5, -40, -24, -26, 19, -29, 31, -3, 14, -6, -17, -19, 8, 24, 6, 20, 17, 3, 0, -2, 31, -57, -63, -47, -22, -51, 23, -59, -27, 20, -64, 2, -41, -35, -69, -48, -20, -47, 8, -80, -49, -27, -53, -30, -30, -15, -89, -61, -36, -3, -76, -68, -41, -69, -69, -44, -71]

Sum of differences: 199750

Spearman's Coefficient from my algorithm: -0.19861986198619852

Scipy's Spearman Coefficient -0.08120983113112498

None

The frequency and word length sorted by frequency:

[(5242, 'to', 2), (5204, 'the', 3), (4897, 'and', 3), (4293, 'of', 2), (3189, 'i', 1), (3130, 'a', 1), (2529, 'it', 2), (2483, 'her', 3), (2400, 'was', 3), (2364, 'she', 3), (2188, 'in', 2), (2151, 'not', 3), (1999, 'you', 3), (1976, 'be', 2), (1811, 'he', 2), (1800, 'that', 4), (1626, 'had', 3), (1441, 'but', 3), (1437, 'as', 2), (1347, 'for', 3), (1321, 'have', 4), (1243, 'is', 2), (1218, 'with', 4), (1212, 'very', 4), (1154, 'mr', 2), (1150, 'his', 3), (1033, 'at', 2), (974, 'so', 2), (848, 'all', 3), (837, 'could', 5)]

The frequent words are as following (shown only 30 words):

[('to', 5242), ('the', 5204), ('and', 4897), ('of', 4293), ('i', 3189), ('a', 3130), ('it', 2529), ('her', 2483), ('was', 2400), ('she', 2364)]