Contents

[Candidate Information: 1](#_Toc36220329)

[Requirements: 2](#_Toc36220330)

[Launching the program: 2](#_Toc36220331)

[Inputs: 2](#_Toc36220332)

[Output: 3](#_Toc36220333)

[Example cmd line script: 3](#_Toc36220334)

[Executive Summary: 3](#_Toc36220335)

[Problem Statement: 3](#_Toc36220336)

[Workflow: 4](#_Toc36220337)

[Structure Chart: 5](#_Toc36220338)

[Implementation: 5](#_Toc36220339)

[Software tools used: 5](#_Toc36220340)

[Step 0 (master): 5](#_Toc36220341)

[Step 1 (arparser\_implementation): 5](#_Toc36220342)

[Step 2 (glob\_implementation): 6](#_Toc36220343)

[Step 3 (frequency\_table): 6](#_Toc36220344)

[Step 4 (interesting\_words): 6](#_Toc36220345)

[Step 5 (split\_sentences): 7](#_Toc36220346)

[Step 6 (output\_configuration): 8](#_Toc36220347)

[Step 7 (clean\_up\_evaluation): 8](#_Toc36220348)

[Results and Performance: 9](#_Toc36220349)

[Conclusion and Further steps: 10](#_Toc36220350)

[Final remarks: 11](#_Toc36220351)

# 

# Candidate Information:

Name: Ujjwal Tamhankar

email: [utamhank1@gmail.com](mailto:utamhank1@gmail.com)

Phone: +1 (856)-952-6794

linkedIn: <https://www.linkedin.com/in/ujjwal-tamhankar/>

github: <https://github.com/utamhank1>

Coursework Portfolio url: <https://drive.google.com/file/d/1_RUJGLZ9wz4e15jQFFcyBCD0lMG5w0uY/view?usp=sharing>

GitHub repository for work done on this assignment:

<https://github.com/utamhank1/nlp_case_study>

# 

# Requirements:

1. The following python 3.7 modules are used in the execution of this file, non-standard library packages are bolded and version number is given:

* python 3.7
* argparse
* os
* sys
* glob
* **pandas (1.0.2)**
* collections
* re
* csv
* string
* **numpy (1.18.1)**

The *requirements.txt* file contains the two non-standard library python modules that need to be installed into the virtual environment.

1. You must have a directory of plain text files that you wish to analyze, the path to this directory cannot contain any spaces or special characters.
2. You must have the following .csv file present in your working directory (NOT the directory which contains the text files to analyze!)

* *stop-word-list.csv*

## Launching the program:

Enter the following into the command line:

*python doc\_parser.py -d [YOUR DIRECTORY] -n [YOUR NUMBER]*

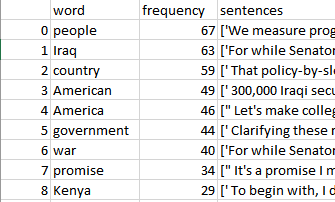
### Inputs:

There are two user-supplied inputs to the program *doc\_parser.py*:

1. -d [directory]: This is the path to the directory which contains the list of .txt files that you want to analyze, it cannot contain any spaces or special characters.
2. -n [number]: This is an integer number that specifies the length of the output.csv file in regards to the number of interesting words that the user desires to see the occurrences and sentences for; for example: if you want to see the top 25 most frequently occurring interesting words, enter 25. This number must be less than 100.

### Output:

There will be an *output.csv* file that will be parsed into the working directory with the following format:



### Example cmd line script:

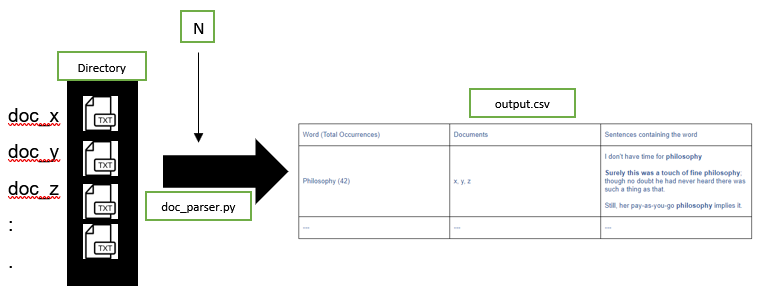
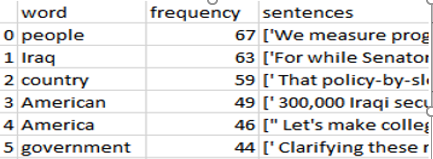
In my case, the script to run the program looks like:

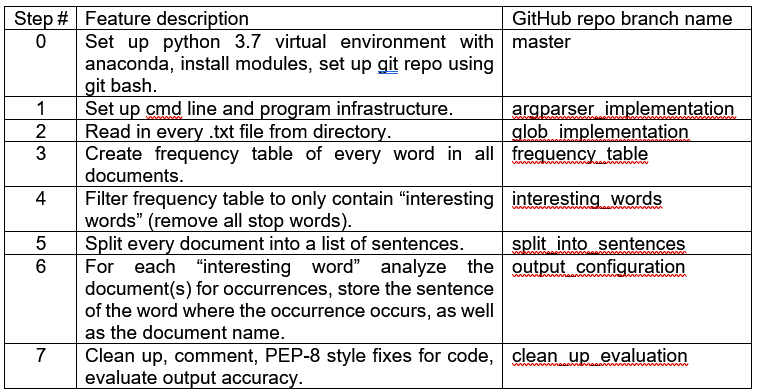
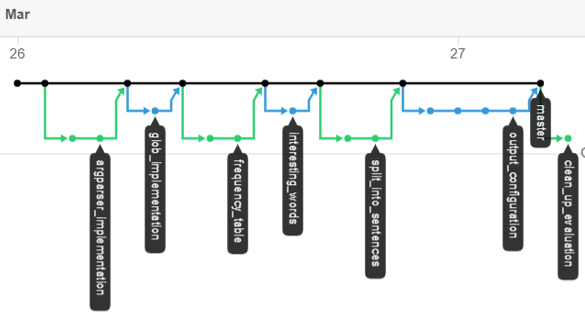
python doc\_parser.py -d C:\Users\Ujjwal\Desktop\Eigen\_files\EigenBETask\EigenTaskExampleDocs\Test\_Docs -n 25

# Executive Summary:

As part of this task I built a program called *doc\_parser.py* that analyzes a directory of text documents, executes the extraction of words, their associated frequencies, the sentences in which the words appear, and the documents in which the words appear.

The program *doc\_parser.py* takes in user inputs for the path to the directory containing the list of .txt files to analyze, and N, which is the N number of most frequent interesting words the user desires to extract the sentences, frequencies and documents for. The program input output structure is represented pictorially as followed:



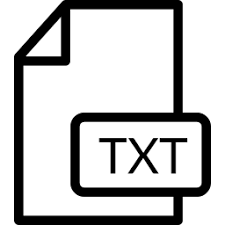
In building the d*oc\_parser.py* program, I took the following iterative steps following the timeline:

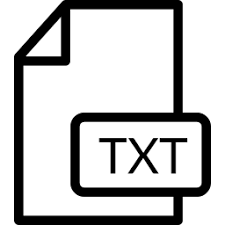
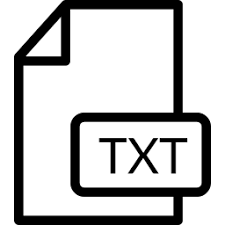
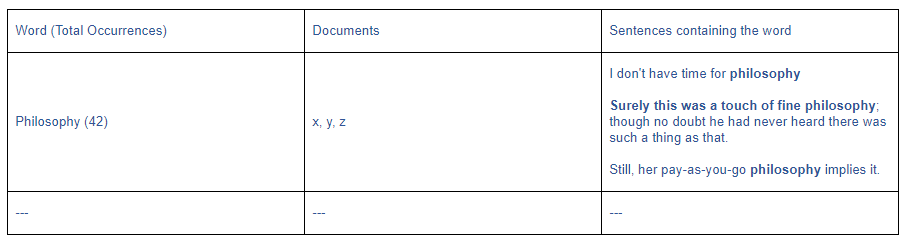
The *output.csv* had an overall **accuracy of** **96.4%.** This accuracy metric was based on the differences between the reported frequency for the word in the output file, and the measured occurrence of that word in the parsed sentences for the first 25 words. In terms of runtime, the program took between 50s and 1m 30s to run.

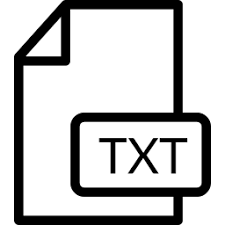
Some improvements to the code could include runtime optimizations, inclusion of additional English language syntactic edge cases, and utilization of the python nlp-specific module spacy (in this case, I did not use it because I was not as familiar with it as I was of pandas and numpy).

# Problem Statement:

In this assignment we are tasked with analyzing a set of text file documents and extracting the most frequently used words within them. Specifically, we want to extract the “most interesting” words, document their occurrences and view the specific documents and sentences in which those words occur in an output similar to the one below:



doc\_x

doc\_y

doc\_z

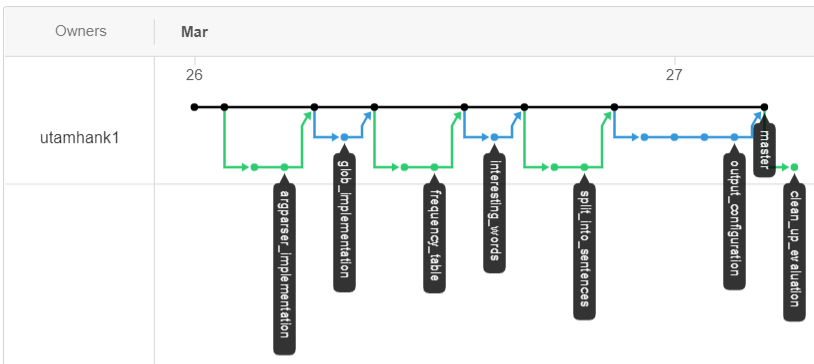
:

# Workflow:

In order to solve this problem, I formulated the following steps, each designed to break up the problem into smaller parts, solve each of them, and iteratively converge on the end solution. Each of these steps has an associated branch within the GitHub repo (<https://github.com/utamhank1/nlp_case_study>) with detailed commit messages showcasing what was done in every step.

|  |  |  |
| --- | --- | --- |
| Step # | Feature description | GitHub repo branch name |
| 0 | Set up python 3.7 virtual environment with anaconda, install modules, set up git repo using git bash. | master |
| 1 | Set up cmd line and program infrastructure. | argparser\_implementation |
| 2 | Read in every .txt file from directory. | glob\_implementation |
| 3 | Create frequency table of every word in all documents. | frequency\_table |
| 4 | Filter frequency table to only contain “interesting words” (remove all stop-words). | interesting\_words |
| 5 | Split every document into a list of sentences. | split\_into\_sentences |
| 6 | For each “interesting word” analyze the document(s) for occurrences, store the sentence of the word where the occurrence occurs, as well as the document name. | output\_configuration |
| 7 | Clean up, comment, PEP-8 style fixes for code, evaluate output accuracy. | clean\_up\_evaluation |

## Timeline:



# Implementation:

## Software tools used:

In doing this project, I used the following software tools:

* Pycharm (Python IDE)
* Anaconda (For virtual environment management)
* GitHub (For version control, pull requests, and project management)
* Git Bash (For creating branches, commits and pulls)
* Windows command line (For running, debugging, testing code)
* Microsoft Excel (Data Validation)
* Microsoft Word (Report Creation)

## Step 0 (master):

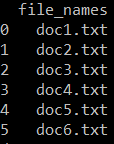
To begin this project, I created a new python 3.7 virtual environment and installed the numpy and pandas modules. Next, I initiated a new repo on GitHub, forked it on my local machine and made initial commits of the project requirements.

## Step 1 (arparser\_implementation):

In order to run this program from the command line, I used the built-in python module argparser to set up inputs and wrote some basic input validation statements. Additionally, I wrote the basic program infrastructure following python best practices.

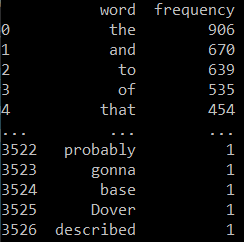
## Step 2 (glob\_implementation):

In this step, I used the glob module to read in files in a directory that follow the same pattern (in this case, end in a .txt filename) one by one. The data structure that was created after this step looked like this:



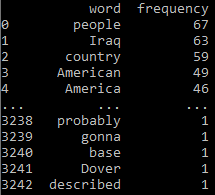
## Step 3 (frequency\_table):

In this step I used the Count function from the collections module to count the number of occurrences of every word in a conglomerated string of all the words of every document. After some cleaning, the data structure parsed at the end of this step looked like this:

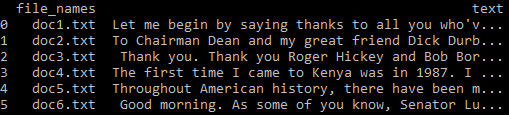


## Step 4 (interesting\_words):

As can be seen with the data structure parsed after step 3, many of the most frequent words have little meaningful value (I learned, that in NLP, these words are referred to as “stop words”). In this step, I import in the file *stop-word-list.csv* which contains a list of these stop words and use it to filter out these words from the freq\_df data structure using a loop. At the end of this step, we get two new data structures: 1) an updated freq\_df minus all the stop words



and 2) a data structure all\_data that contains the filenames and all the associated text that goes with that file name.



## Step 5 (split\_sentences):

In this step, I split every document into its constituent sentences initially stored in a python dict called docDict with the format shown below:

{ “0” : [sentence 1, sentence 2, …]

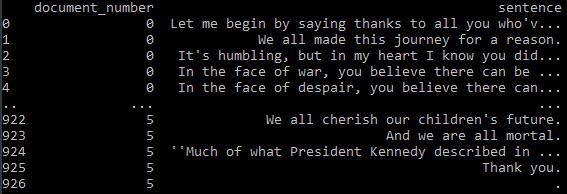
“1” : [sentence 1, sentence 2, …]

“2” : [sentence 1, sentence 2, …]

“3” : [sentence 1, sentence 2, …]

…}

I then transformed this data to fit into a pandas dataframe called split\_sentences with the format shown below:



This had the advantage of giving each sentence in every document its own unique identifier and mapping it to the document that it originated from.

## Step 6 (output\_configuration):

In this step, I analyzed every one of the data structures created from the steps prior and used a for loop to construct the final output data structure by integrating pieces, indexes and columns from each of the data structures created earlier. The pseudo-code for this for loop looked like (with N being the user-supplied value for number of words to analyze):

output = dataframe with (columns = [‘word’, ‘frequency’, ‘document\_names’, ’sentences’])

for i in range (0, N):

create empty list to hold names of documents

create empty list to hold names of sentences that contain the word in question

for j in range(0, len(split\_sentences)):

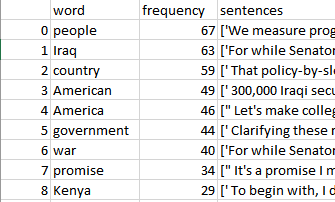
if (word from frequency table is in the sentence):

add that document name that contained the sentence to the list of doc names

add that sentence to the list of all sentences that contain the word

attach unique entries of the list of the names of the documents and sentences to output dataframe

After implementing this pseudocode, I obtained the output table below for the first 25 most interesting words (8 shown):



## 

## Step 7 (clean\_up\_evaluation):

In this step I use excel and MS word to perform some data validation and review the accuracy of the model. I also added comments and PEP-8 coding style fixes

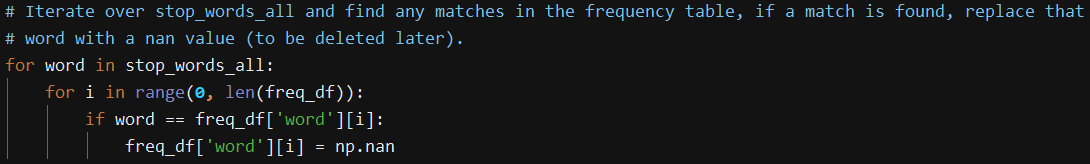
# Results and Performance:

In earlier iterations of *output.csv* it was found that there were some large discrepancies between the frequencies of the words that were reported and the actual amount of the words present in the sentences (CTRL-F in MS word tells you how many of the instances there actually are). Upon learning of these I modified the code to 1) not make a distinction between capitalization of the words in sentences, and 2) instead of comparing to see if a piece of a string is within another string, which caused some problems due to nuances/variations of the word and presence of punctuation and other operators, built a helper *function findWholeWord()* that uses the *re.compile()* function and does its best to try and find the actual word within a string (it is not perfect, but significantly better than the approach I used before). By using these two corrections, I was able to boost the accuracy of the calculated word frequency to the measured word frequency from 83.1% to 96.4% (weighted by the number of occurrences) as seen in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| Entry Number | Calculated word frequency | Measured word frequency | % Difference |
| 0 | 67 | 68 | 1.492537 |
| 1 | 63 | 64 | 1.587302 |
| 2 | 59 | 60 | 1.694915 |
| 3 | 49 | 50 | 2.040816 |
| 4 | 46 | 46 | 0 |
| 5 | 44 | 45 | 2.272727 |
| 6 | 40 | 42 | 5 |
| 7 | 34 | 34 | 0 |
| 8 | 29 | 31 | 6.896552 |
| 9 | 29 | 29 | 0 |
| 10 | 28 | 32 | 14.28571 |
| 11 | 27 | 27 | 0 |
| 12 | 25 | 25 | 0 |
| 13 | 25 | 25 | 0 |
| 14 | 23 | 24 | 4.347826 |
| 15 | 23 | 23 | 0 |
| 16 | 21 | 21 | 0 |
| 17 | 19 | 19 | 0 |
| 18 | 19 | 21 | 10.52632 |
| 19 | 18 | 20 | 11.11111 |
| 20 | 18 | 22 | 22.22222 |
| 21 | 18 | 21 | 16.66667 |
| 22 | 17 | 17 | 0 |
| 23 | 17 | 17 | 0 |
| 24 | 16 | 19 | 18.75 |
| Sum/Weighted Avg % Error | 774 | 802 | **3.617571** |
| Avg % Error |  |  | **4.755788** |

As we can see, 100% - 3.617% = 96.4% in terms of accuracy. I deemed this an acceptable amount of error, especially given the nuances of language are generally hard to encapsulate with the concrete mathematical rules that most programming languages follow.

In terms of runtime, the program does not take a trivial time to run (on my machine it was between 50s and 1m 30s, depending on number of background processes) as a peruse through all of the nested for loops tells me that the highest order operation within the program is of degree O(N^2) which for a dataset size in this domain is not ideal, but acceptable as long as N does not grow exponentially large as well. The longest running piece of code, in this case, is attributed to the comparison of every word in stop\_words to every word frequency in freq\_df which is O(N^2); the snippet of code of which is shown below:



# Conclusion and Further steps:

In this task I was able to extract words from a text file, determine their frequency in a given number of documents, and extract the sentences in which these words appeared and map those to the exact document(s) those words appeared in. Of course, there is still room for improvement. Language, by its very nature, has many nuances, colloquialisms and rules that computer programs are just starting to be able to decrypt.

In my program, there are still small errors and edge cases that would have to be resolved to get the accuracy of the model closer to 100%. Further, the text file data is not perfect, there are still many special characters and strings that were incorrectly parsed by the program, (a cursory peruse shows the occasional appearance of wingding-like characters randomly dispersed in the text). In the context of this assignment, the law of diminishing returns applies as, in my opinion, the incremental gain in accuracy does not justify the amount of effort needed to map out every single edge case and possible word appearance scenario. However, in more sensitive applications, it might be worth investing in a better way of handling these.

In terms of program runtime, potential improvements could be made by excluding the stop-words when reading in the text files when creating the frequency table, instead of reading all of the words, counting the frequencies and then removing the stop-words from the frequency list as the program currently does. Of course, this would take some additional effort on the other end when recreating the sentences (since they would have been originally read in without stop-words!). Further improvements could be made by keeping the data structures as native python lists or dictionaries instead of converting them to pandas dataframe or numpy lists (less overhead of modules to import).

## Final remarks:

I would like to thank Ms. Markov for her time in interviewing me for this role and for answering my questions about Eigen technologies. I had a lot of fun working through this problem and learned quite a lot about NLP. I primarily used the pandas and numpy libraries in doing this project because I was more familiar with them, but if I were to do a project like this again, I would dig into the documentation for the spacy python module that I discovered while researching nlp stop-word datasets and use some of its functionalities to further optimize the accuracy of the output.

## 