Artificial Intelligence Project: POS Tagging

September 28, 2020

Ritik Jain 18114068
Priyanshu Garg 181140058
Yugantar Arya 18114084

Contents

1 Week 1 **i**
1.1 Objective i
1.2 Procedure i
1.2.1 Phase 1 Filtration i
1.2.2 Phase 2 Collection i
1.2.3 Phase 3 Verification i
1.3 Code Screenshots ii
1.4 Execution v
1.5 Output Files vi
1.6 Results ix

2 Week 2 **x**
2.1 Objective x
2.2 Procedure x
2.2.1 Phase 1: Construction x
2.2.2 Phase 2: Serialization x
2.3 Code Screenshots xi
2.4 Execution xiii
2.5 Output Files xiv
2.6 Results xv

1 Week 1

Date: September 28, 2020

1.1 Objective

To filter the text by extracting word and POS tag for the word

1.2 Procedure

The program does its job in 3 phases. The number of phases could have been reduced to 1 but I went with this code since it doesn't take much time to pre-process all the files and it has to be done only once; hence I preferred clarity of code over the small extra time it takes.

1.2.1 Phase 1 Filtration

The program processess all the raw xml files and converts them into filtered files The program makes use of the regex, which is the heart of the program:

<\s*w[^>]*pos="([A-Za-z]*)"[^>]*>\s*([^<\s]*)\s*<\s*/w\s*>

The above regex is a flexible one. It could have been made stricter, given the consistency of the dataset, but I still went with the flexible version since it is more general. A filtered file contains all the word_tag combinations for its xml file. The word,tag combinations are stored in a case sensitive manner.

1.2.2 Phase 2 Collection

The program then collects all the filtered files and assembles them into 1 big collect file. The collect files for Test and Train dataset are made separately

1.2.3 Phase 3 Verification

The verification phase verifies that the filtered data matches the raw data. This is accomplished by re-scanning the raw data with a simpler regex **<\s*/\s*w\s*>** (only for </w> tags) and maintaining the total count of word,tag combinations for each file. The filtered files are also re-scanned, but the regex used is **;**. After the scan is complete, the verification ratio and the accuracy are displayed. If the accuracy is more than 95%, the verification could be assumed to be successful.

After the successful completion of all the phases, we obtain the processed data.

1.3 Code Screenshots

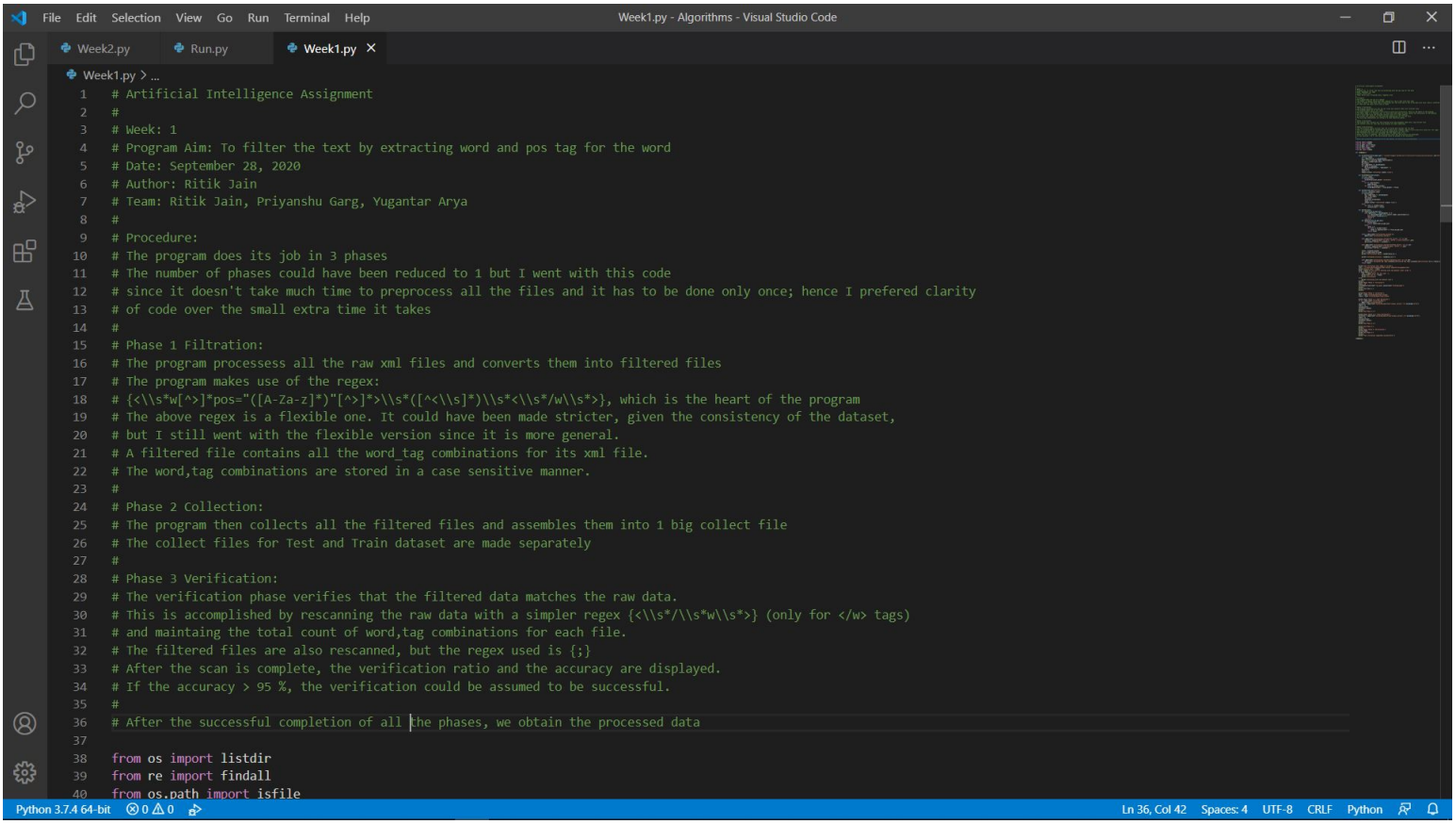

Figure 1: Week-1 File: Section 1

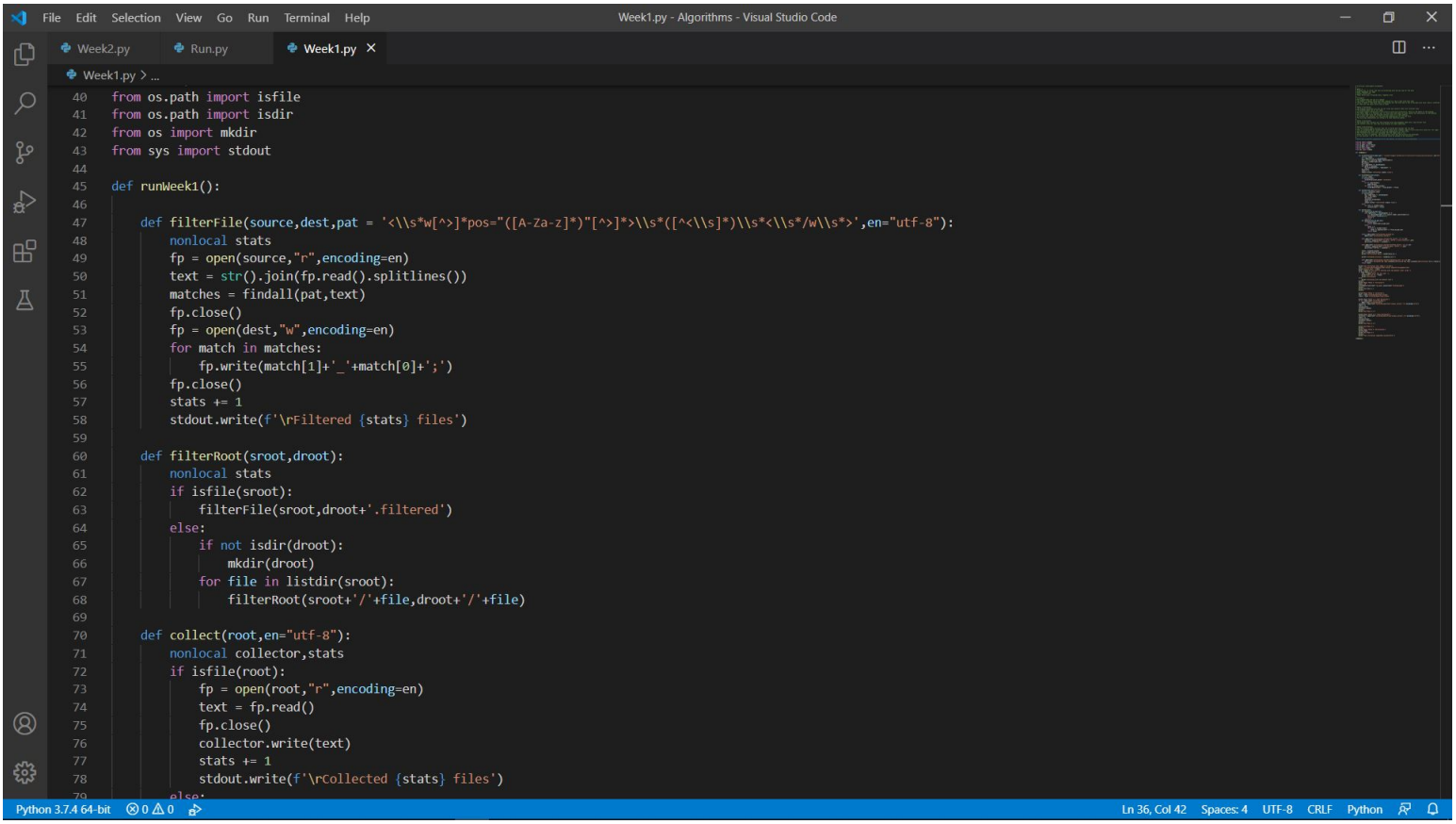

Figure 2: Week-1 File: Section 2

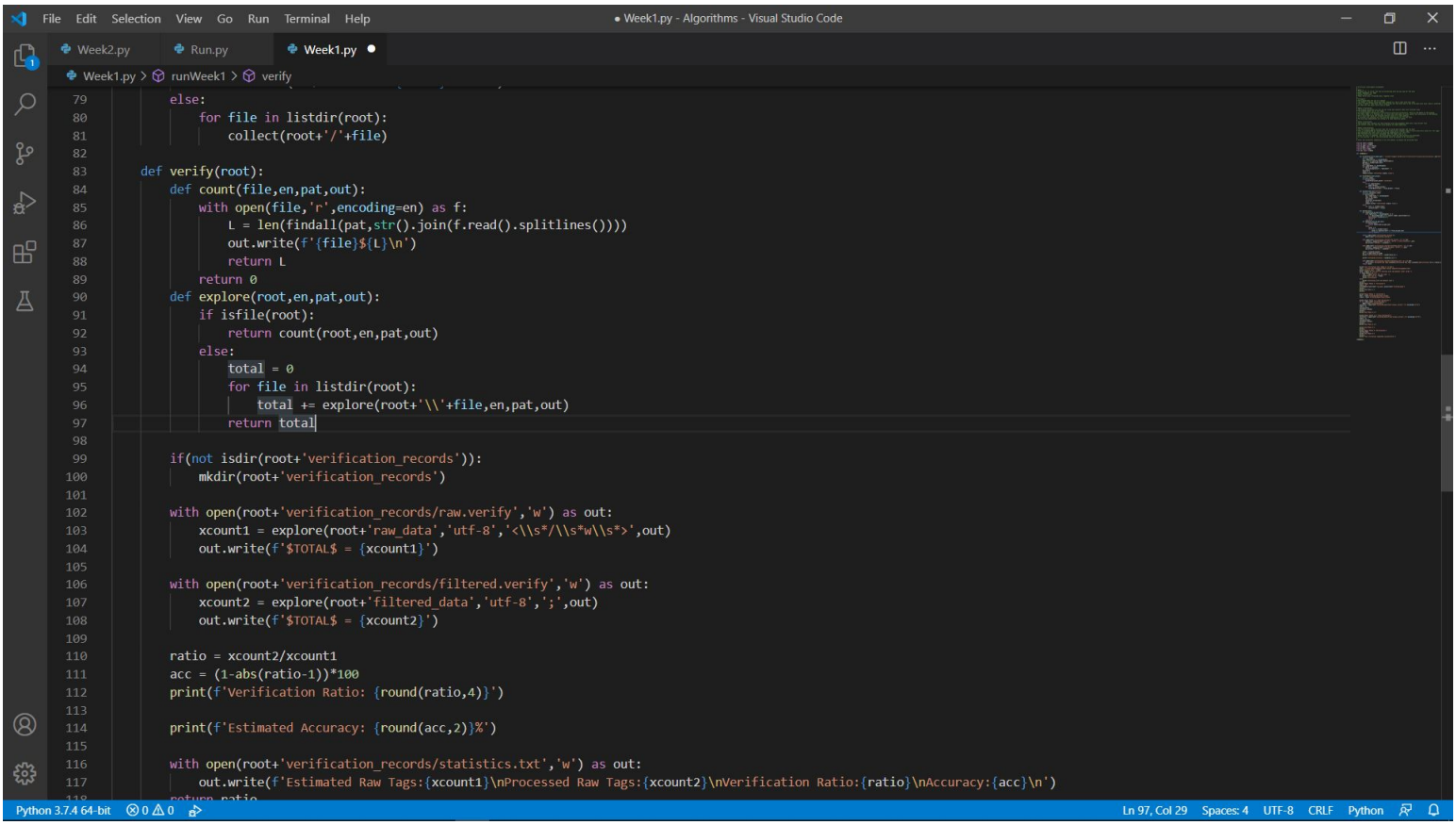

Figure 3: Week-1 File: Section 3

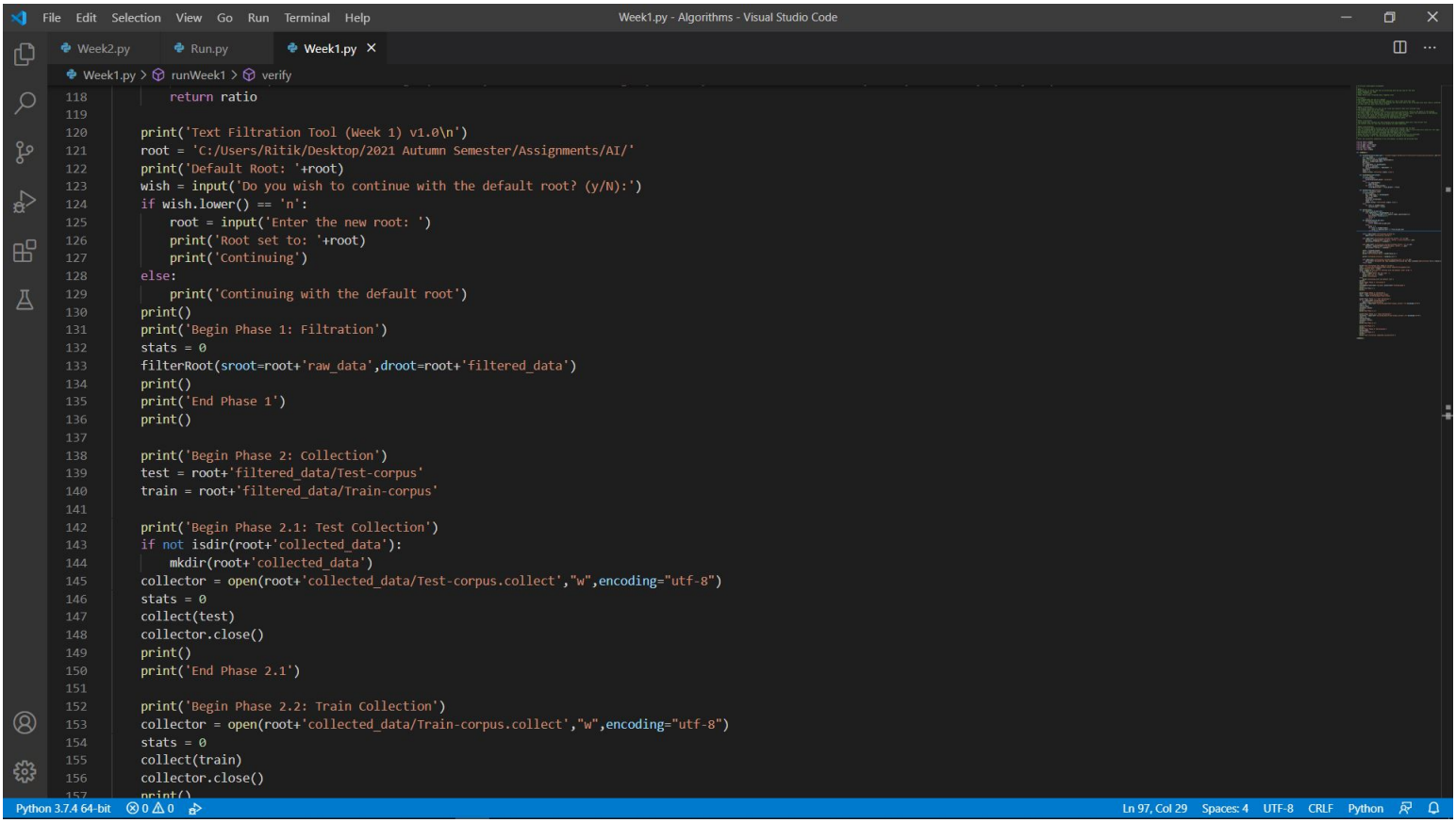

Figure 4: Week-1 File: Section 4

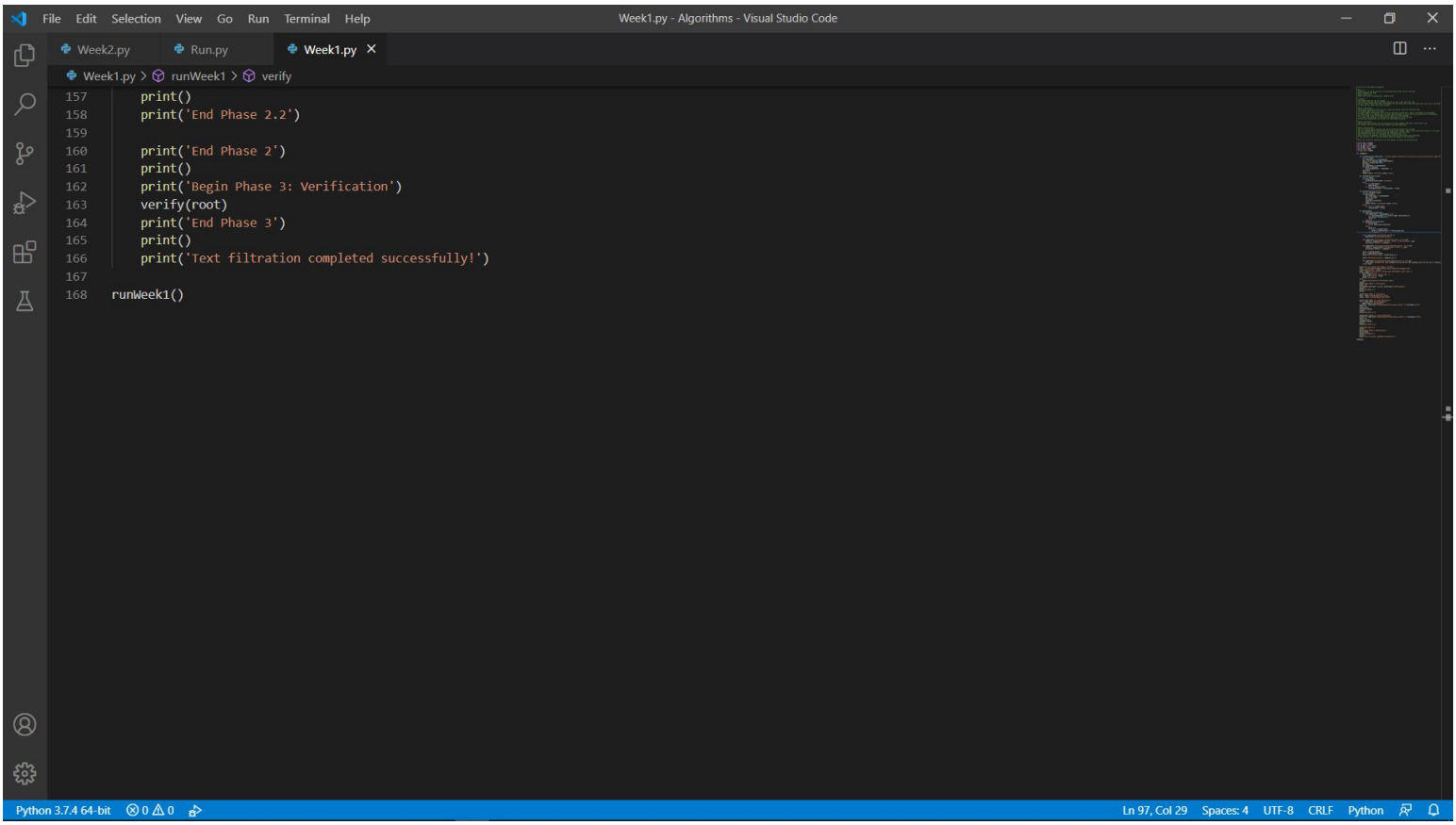

Figure 5: Week-1 File: Section 5

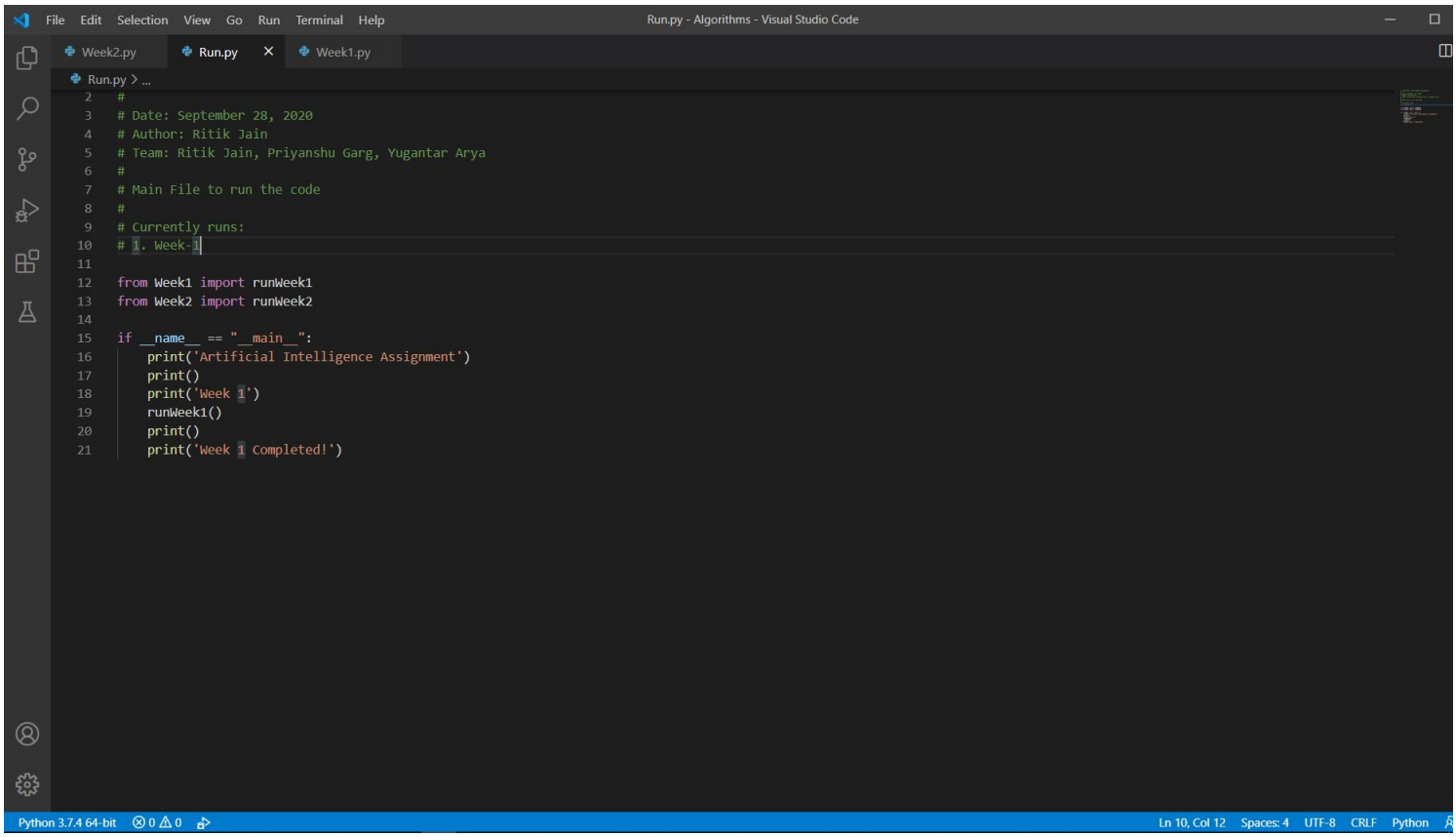

Figure 6: Main File: Run.py

1.4 Execution

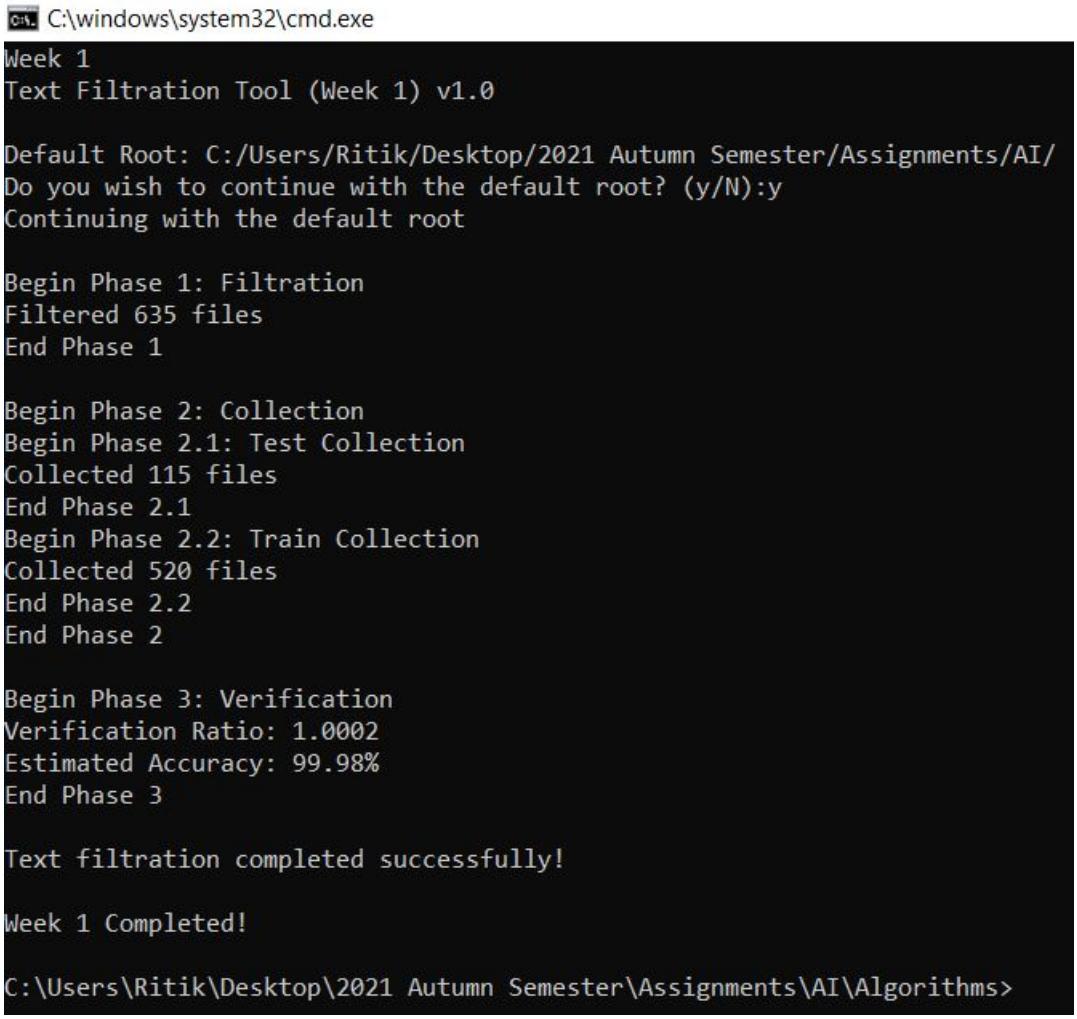

Figure 7: Execution: Run.py

1.5 Output Files

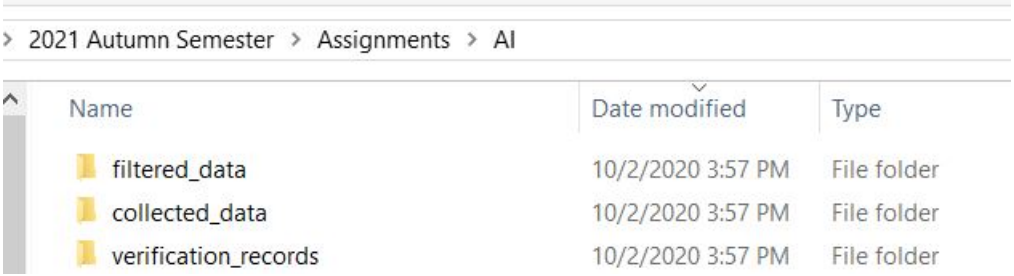

Figure 8: Generated Folders

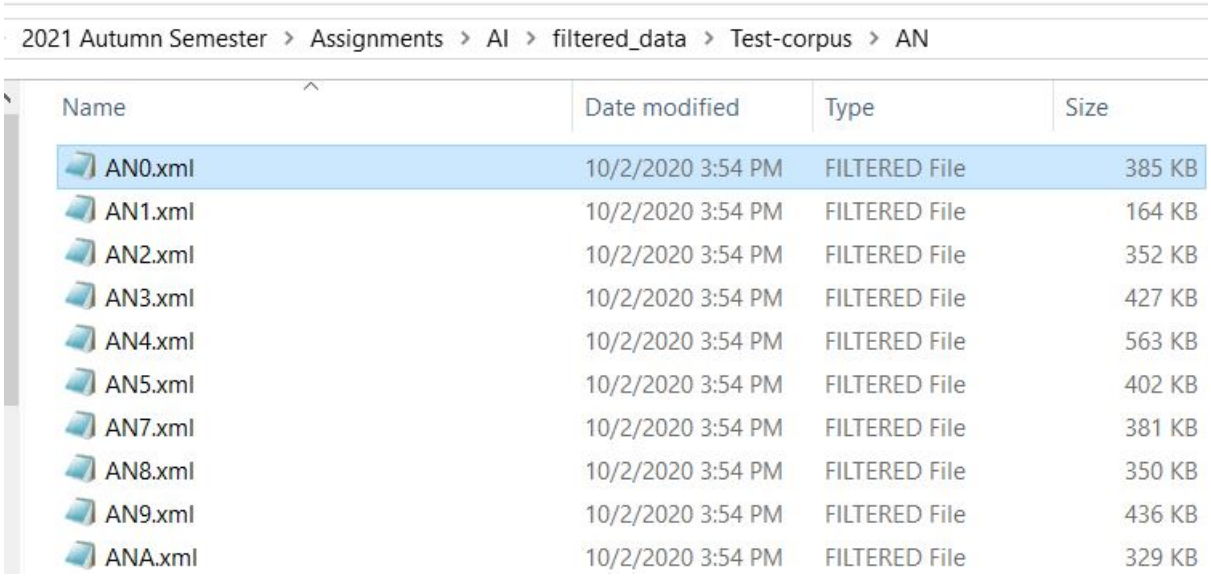

Figure 9: Phase 1 Filtered Files

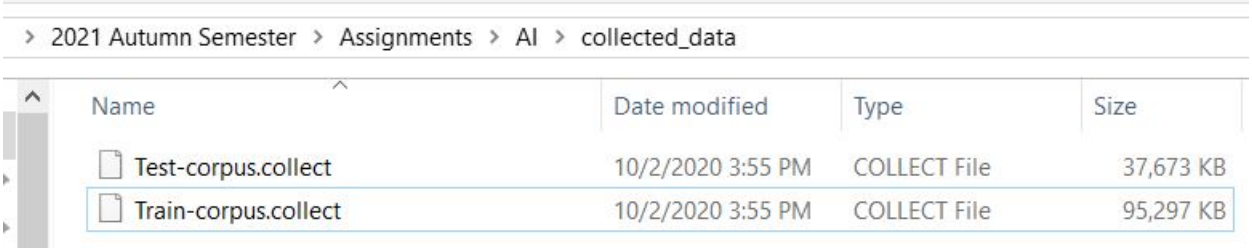

Figure 10: Phase 2 Collected Files

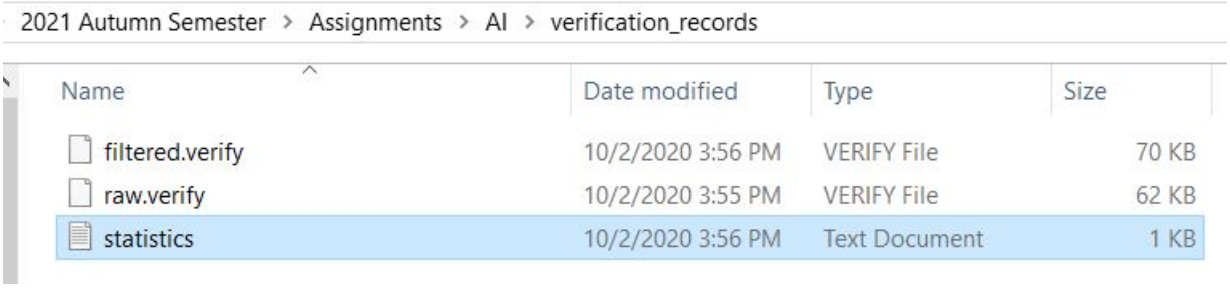

Figure 11: Phase 3 Verification Files

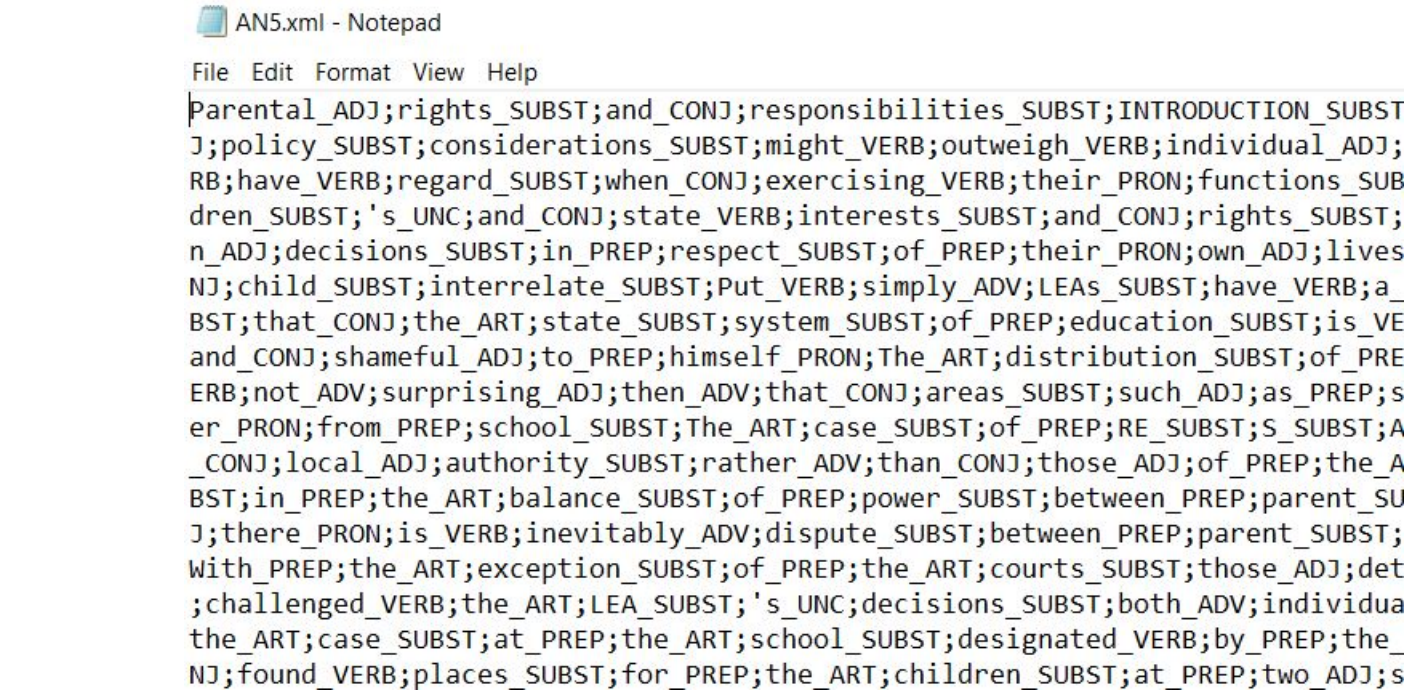

Figure 12: A filtered File

Figure 13: A collected File

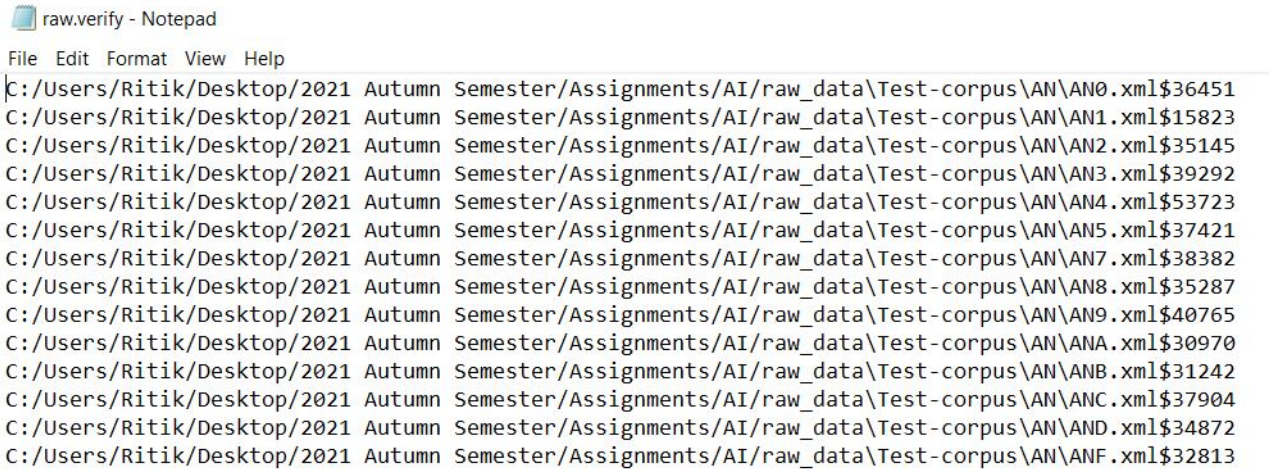

Figure 14: Raw Data Verification File

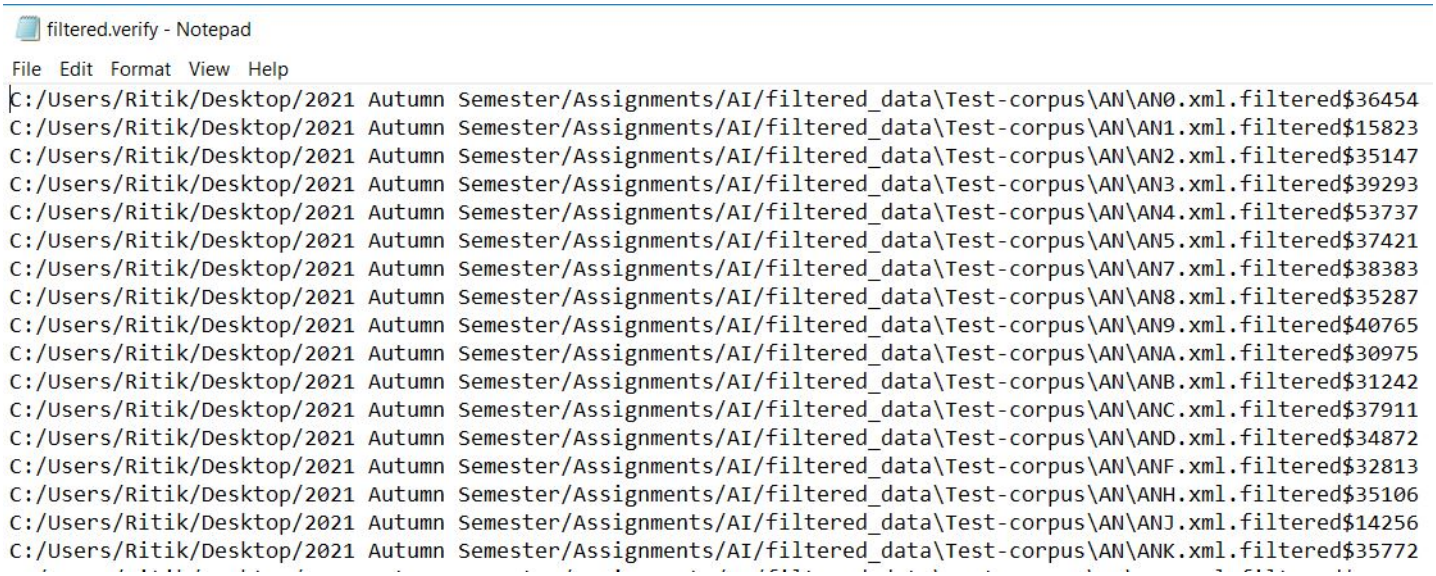

Figure 15: Filtered Data Verification File

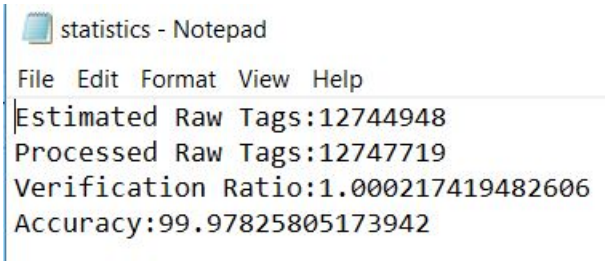

Figure 16: Statistics File

1.6 Results

The raw data was processed with an estimated accuracy of 99.98% or more. We could have made the data more accurate (to about 100%) by improving the regex for scanning raw files, but it would simply not be worth the trouble of investigating the patterns in the huge amount of data, only to correct an error of 0.02% or less, which would hardly affect the accuracy of the application. This data is now ready to be converted into a frequency dictionary in week 2.

2 Week 2

Date: October 2, 2020

2.1 Objective

To create a dictionary which maps word,tag to its frequency

2.2 Procedure

The application works in two phases. The first phase involves the construction of a human-readable dictionary files from both the collected test and train files. The second phase involves the construction of a serialized dictionary file for fast-access.

2.2.1 Phase 1: Construction

Construction of the dictionary in file The collected files are used to create a dictionary (a hashmap) that maps a pair (word, tag) to frequency. The dictionary is stored in a human readable format in a dict file and as a hashmap in the memory.

2.2.2 Phase 2: Serialization

Construction of the serialized dictionary for fast loading This phase involves the serialization of the hashmap in memory using pickle, so that whenever the dictionary is required, it could be loaded rapidly in to the memory from disk.

After the successful execution of the program, we get the required dictionary. Verification phase is not required since it was already done in week 1.

2.3 Code Screenshots

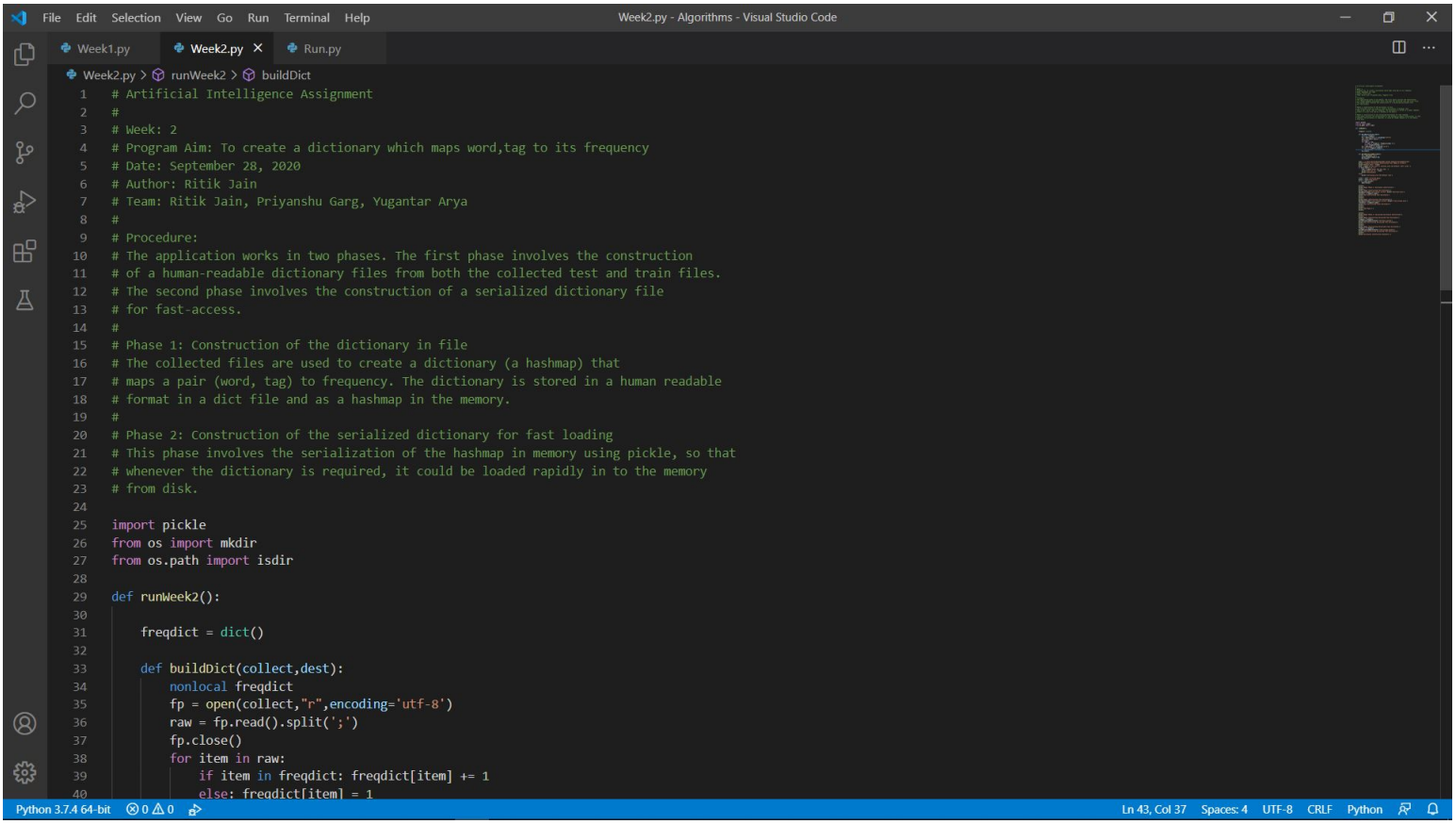

Figure 17: Week-2 File: Section 1

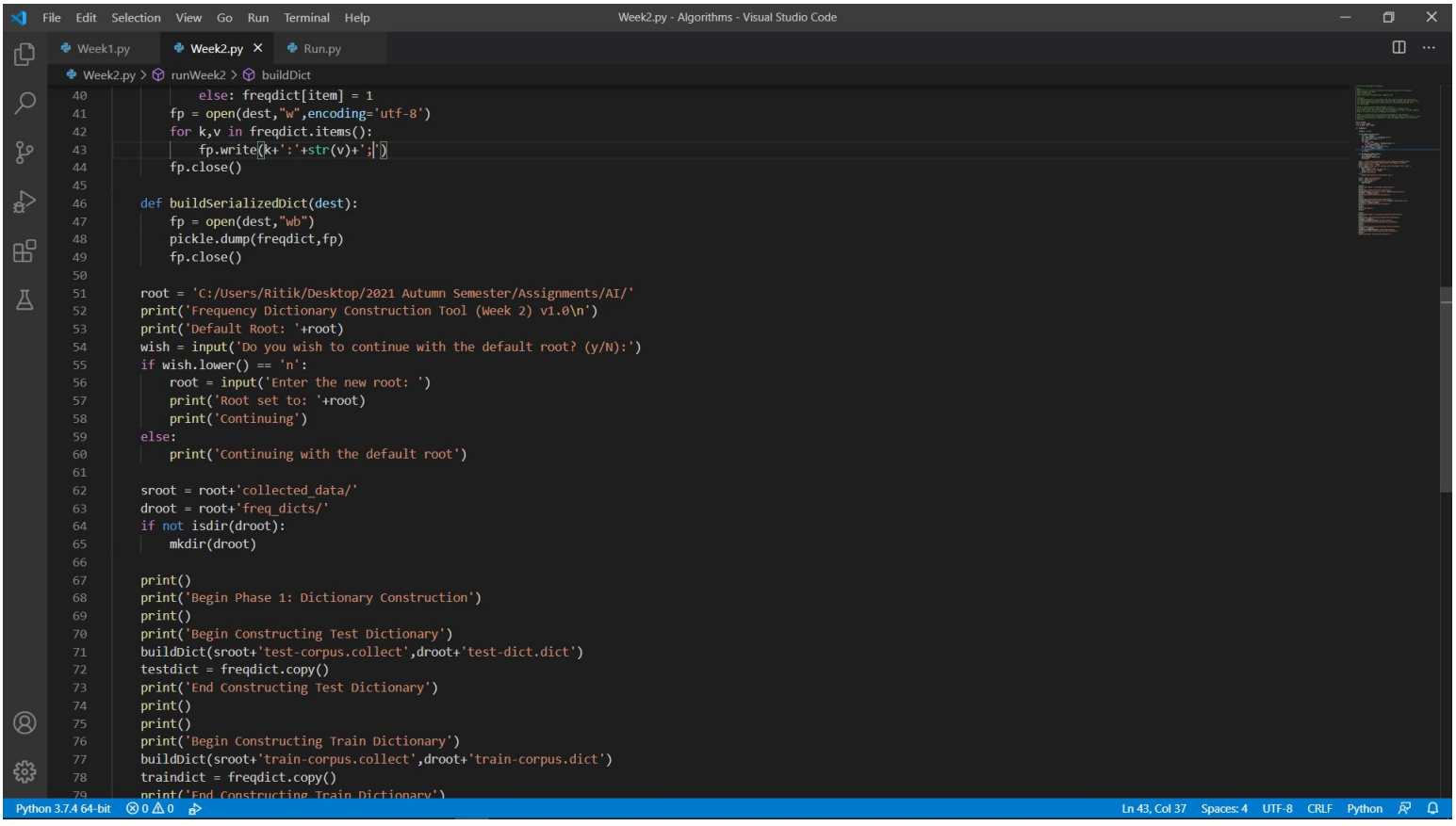

Figure 18: Week-2 File: Section 2

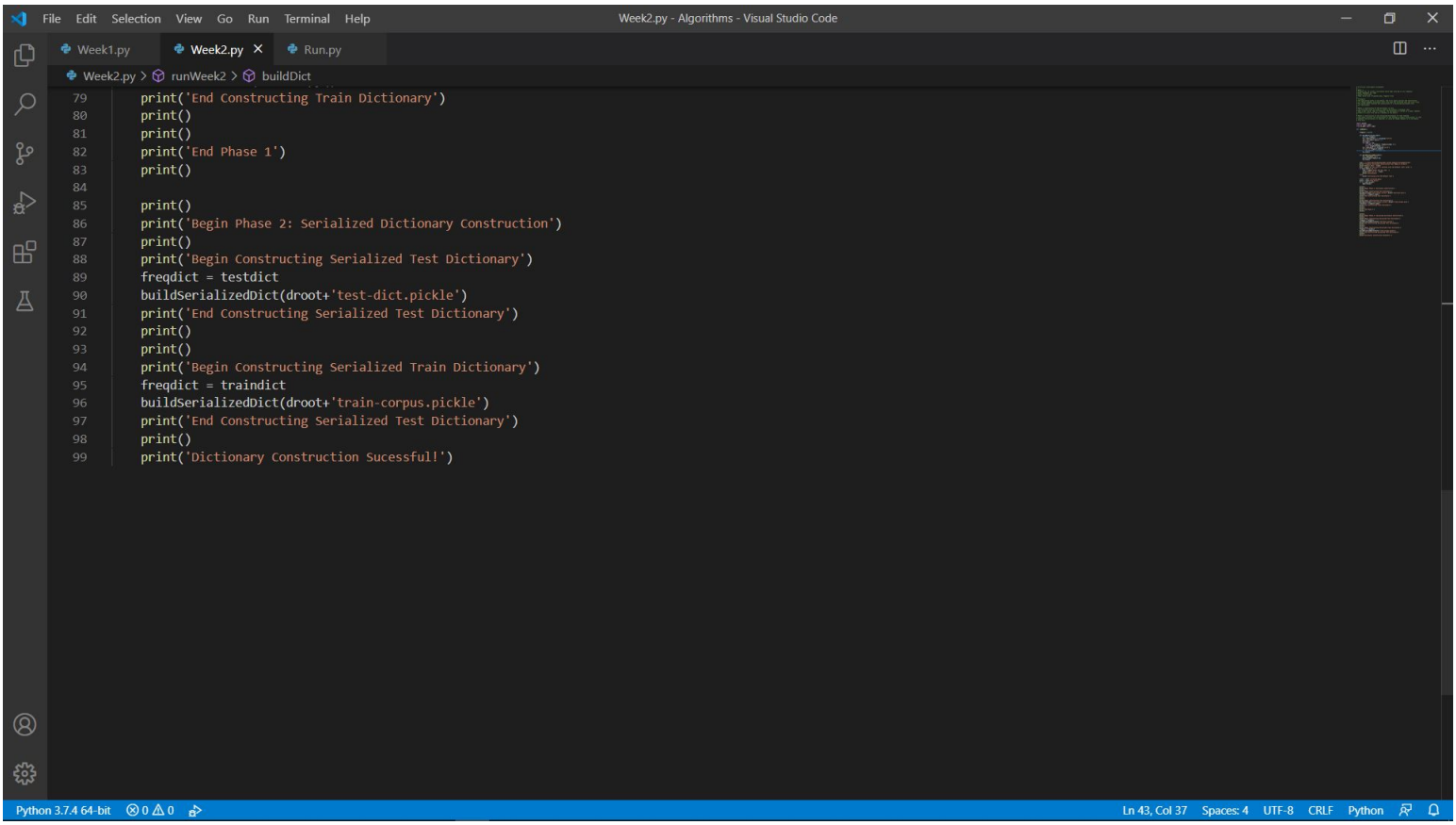

Figure 19: Week-2 File: Section 3

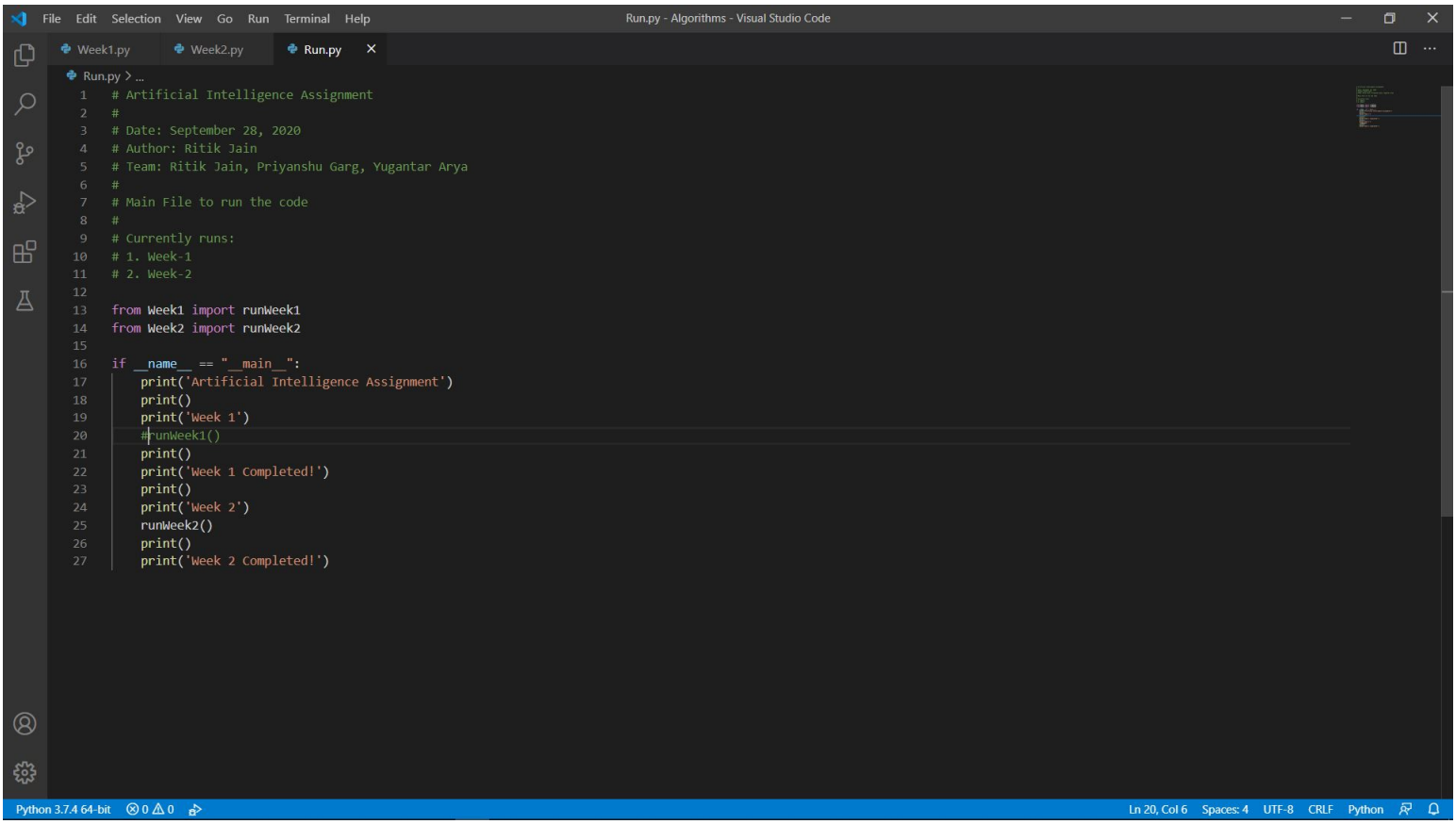

Figure 20: Main File: Run.py

2.4 Execution

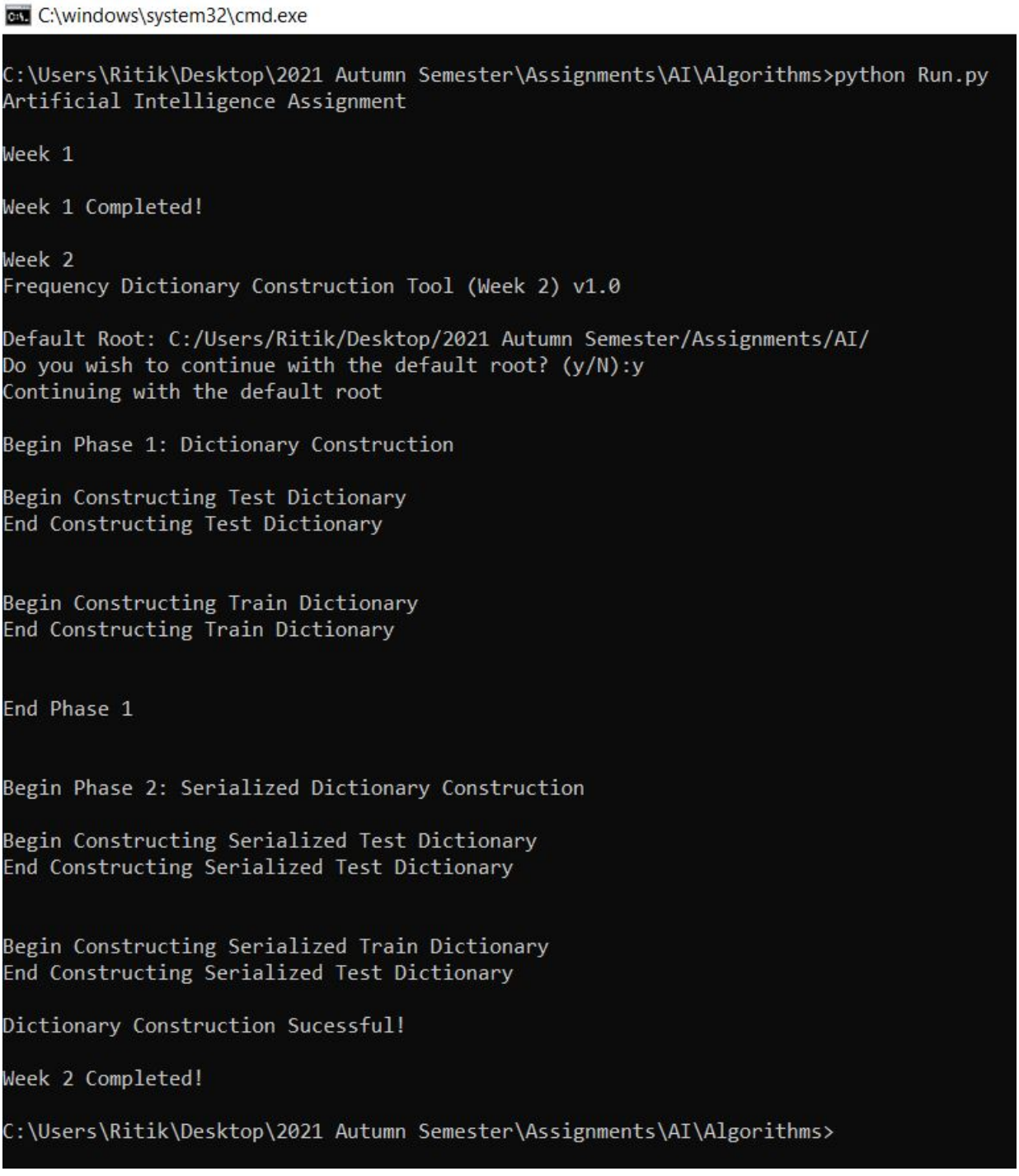

Figure 21: Execution: Run.py

2.5 Output Files

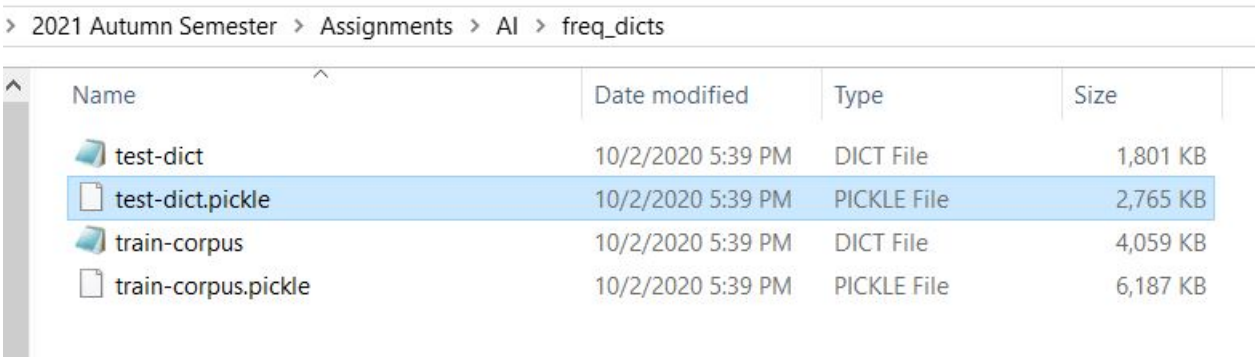

Figure 22: Generated Dictionaries

Figure 23: A section of test dictionary

2.6 Results

The required dictionaries were successfully generated, with the same accuracy as that of week 1. The next step is to analyze the distribution of words and tags, which would be done in the next week.