**DOCUMENTATION**

Below is the explanation of the project. Each file is listed and explained with snapshot of the code

**Task 1. File “task1.py”**

1. **Clean text:**

Text

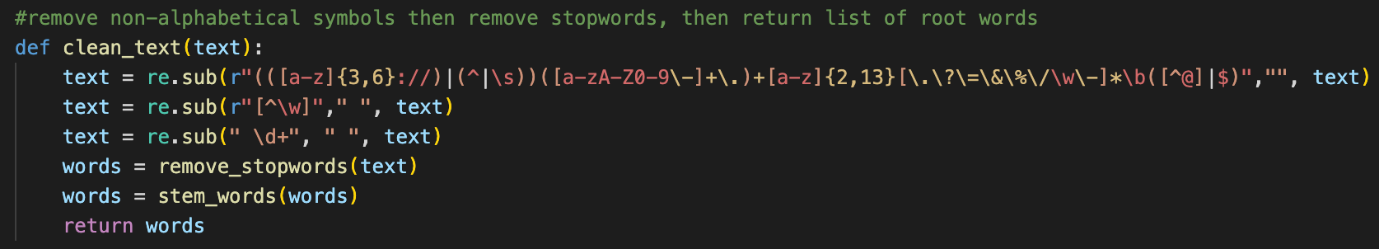
Description automatically generated

* Tokenize the text and then remove stop words.

Graphical user interface, text, application

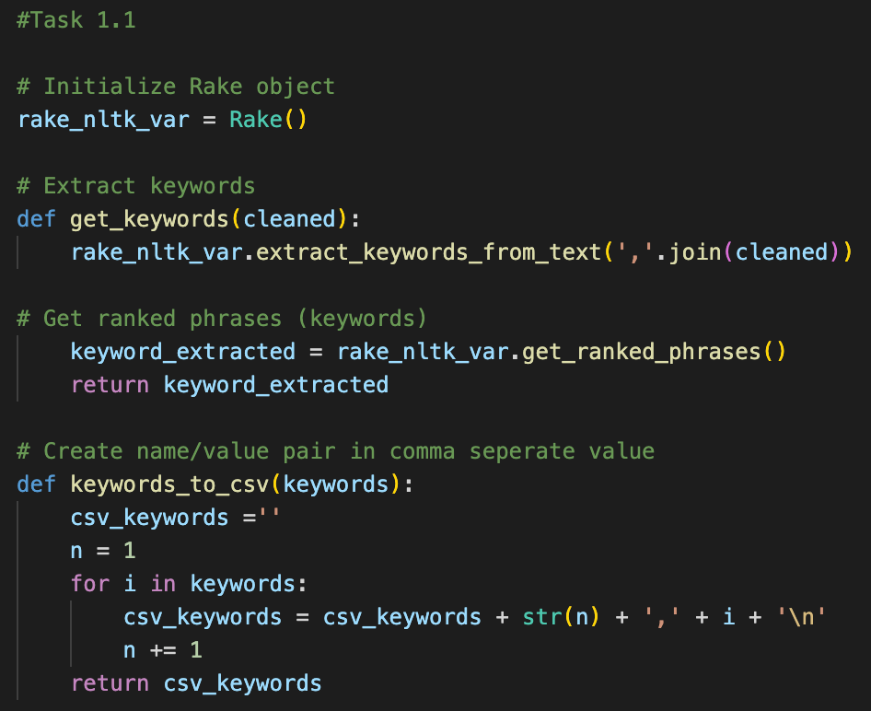
Description automatically generated

* Transform the words into its root form.



* Clean non-alphabetical characters, url, numbers, remove stop words, transform words into root form and then return the cleaned text.

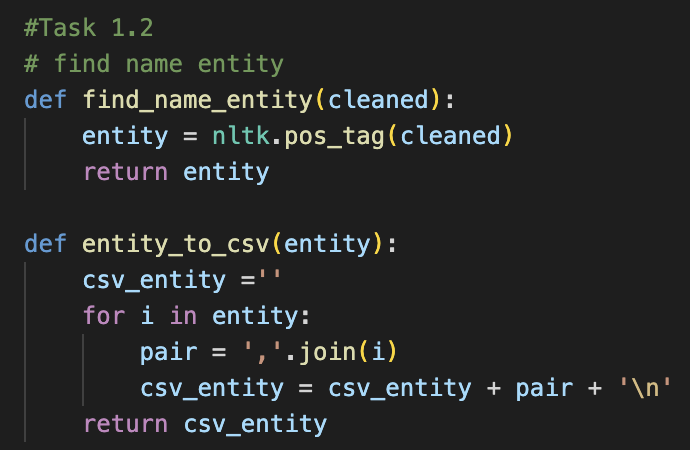
1. **Extract keywords**



* Use nltk to extract keywords from the text.
* Get the rank list.
* Make name/value pair for each key word and its ranking.

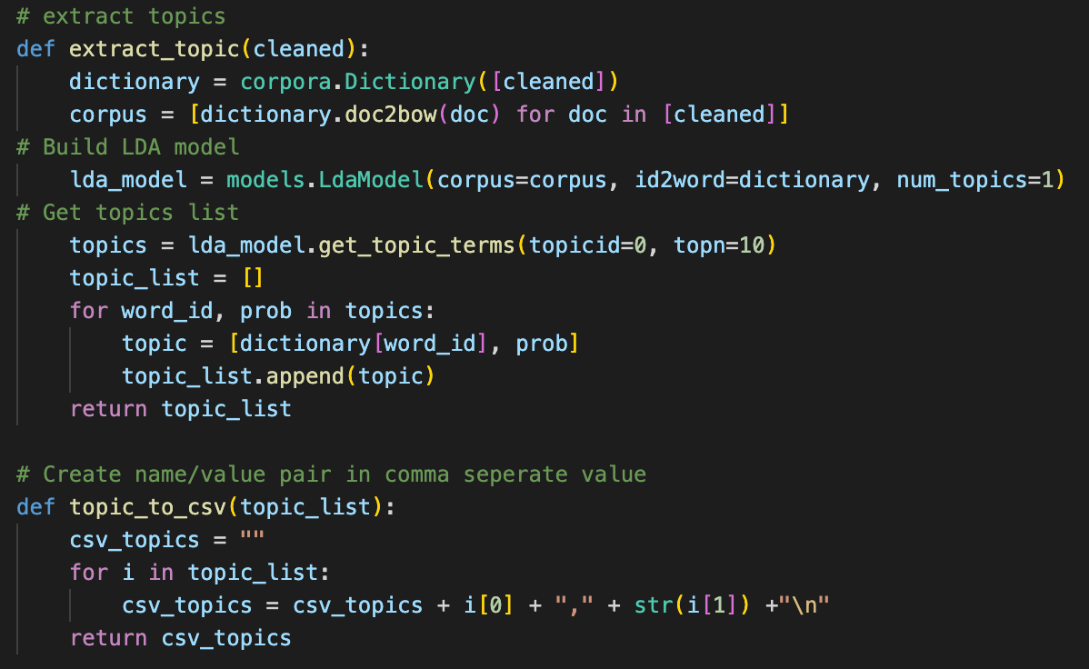
1. **Find name entity**

* Using nltk.pos\_tag, check each word’s name entity.
* Return name/value pair for each word and its name entity.



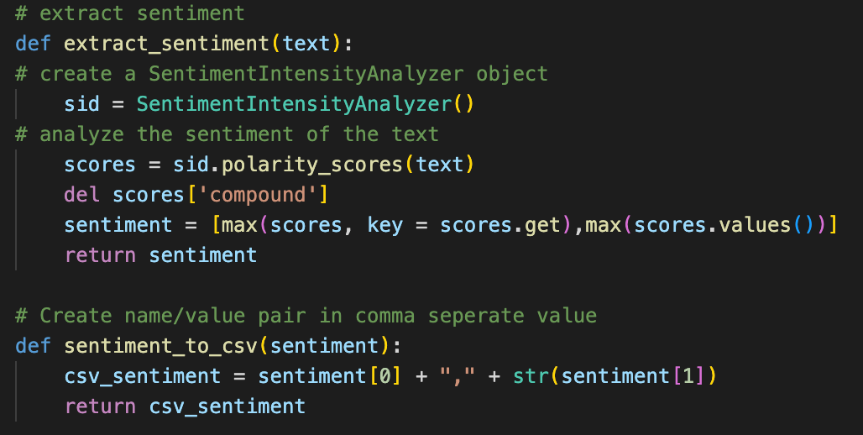
1. **Extract topics**

* Create a gensim dictionary object using the corpus.
* Create a variable in which we store the Bag-of-Words transformed documents.
* Train the model and get the topic list.
* Create name value pair for the topic and the probability.



1. **Extract sentimental.**

* Create SentimentIntensityAnalyzer object.
* Get the sentiment of the text, delete compound value.
* Get the highest score as the final sentiment of the text.
* Create a name/value pair of the sentiment of the text



1. **Connect to tweets dataset and apply all the methods.**



1. **Extract txt. File “txt.py”**

* For task 2, 3, 4, 5, we need txt file extracted from the tweets as the input. Therefore, I’ve made an application to extract all the text needed and write into separate txt file.
* “task2.txt” include the text of each tweet in a line.

Text

Description automatically generated

* “task3.txt” extract the city from the tweet’s ‘address’ field. For tweets without location info, it will return “No location info”. For tweets in countries other than Australia, it will return “Other country”. For tweets without city info, it will return “No city info”. For anything else, return the city name.

Text

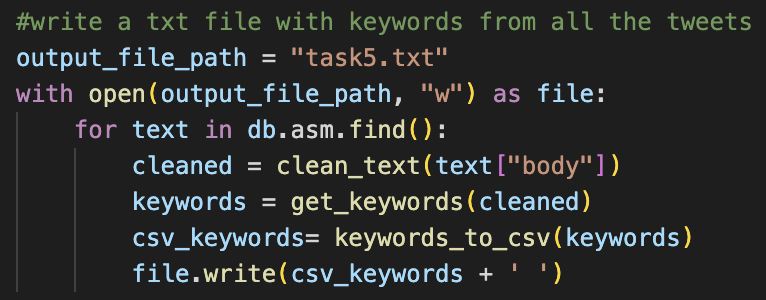
Description automatically generated

* “task4.txt” extract the tweet’s id info numbers, clean all the text and the tweet’s content. Output in comma separated format.

Text

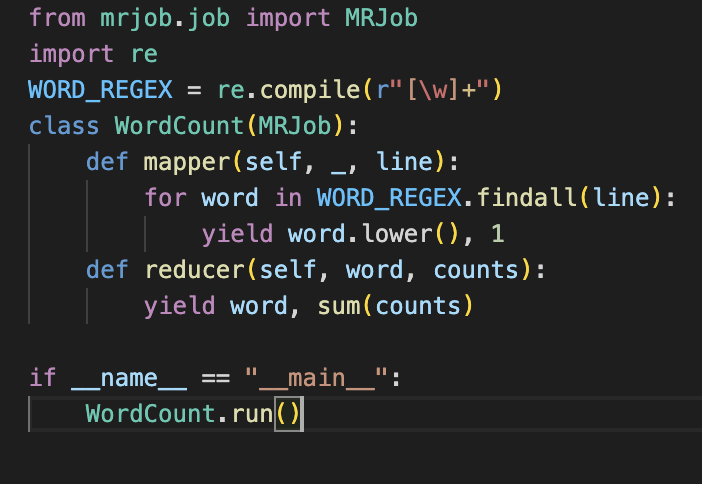
Description automatically generated

* “task5.txt” extract the keywords from each text and output into a string.



**Task 2. File “task2.py”**

* Take input “task2.txt”.
* Split each line into words.
* Map each word with value 1.
* Group same words and sum up the number.



**-Pseudocode:**

Input the txt file

Mapper :

For each word in the line:

Return the word with value = 1

Reducer:

Map all the same word together, sum the value

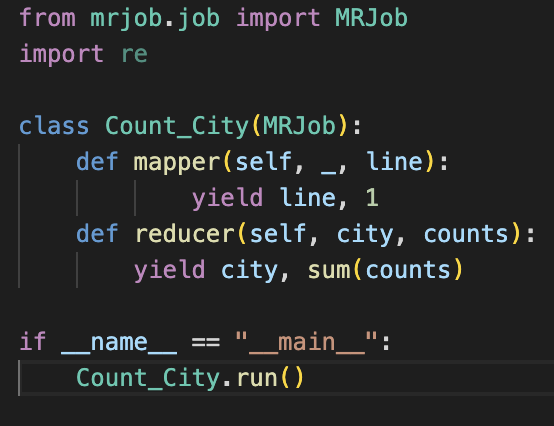
Flowchart:

Diagram

Description automatically generated

**Task3. File “task3.py”**

* Take input “task3.txt”.
* Map each line(city) with value 1.
* Group same cities and sum up the number.



**-Pseudocode:**

Input the txt file

Mapper :

For each line:

Return the line with value = 1

Reducer:

Map all the same line together, sum the value

Diagram

Description automatically generated

**Task4. File ”task4.py”**

**Text

Description automatically generated**

* Take input “task4.txt”.
* Split each line into ID number and tweet’s content.
* Map ID and text into the same key.
* Sort each key by the ID number.

**-Pseudocode:**

Input the txt file

Mapper :

For each line:

Split into id and text.

Return key with value (id, text).

Reducer:

Map all the line together, sort by ID number

-**Flowchart:**

Diagram

Description automatically generated

**File “task4\_2.py”**

* I’ve used the mergesort algorithm to sort the ID.
* Split the tweets into half until it become single tweets, then iterating sort them in order and merge into one

For the performance evaluation, I’ve added the running time for each method.

* MapReduce method increased the running speed by a lot, from 0.015 seconds to 7.9\*10-6 seconds.

**Task5. File “task5.py”**

Text

Description automatically generated

* For the task 5, I’ve used the sklearn library.
* Transforming the text into a matrix of TF-IDF features
* Converting the first row of the TF-IDF matrix into a DataFrame
* Convert the pandas dataframe into DASK dataframe, with n participant = 100 ( split into 100 part to compute )
* Sorting it by descending order of TF-IDF scores and takes n top value ( n is the input number)
* Compute the task

Result:

* Result will be in the csv file with “|” separator.
* N top result display with n as an input number
* Below is the sample result of 6 top words:

Text

Description automatically generated

Limitation:

* DASK is only applied in the sorting step, as I’m using the sklearn library to compute TF-IDF values.

Improvement:

* Perhaps using another method might enable me to find other steps to parallelize.