**Sentiment Analysis on Financial Reports using Python**

This project performs textual analysis on the financial reports to drive sentimental opinion, sentiment scores, readability etc.

The project is categorized into 3 Parts:

1. Data Extraction
2. Data Parsing
3. Textual Analysis

Data Extraction is done by extracting data from links containing SEC EDGAR financial reports.

Data Parsing involves organizing the extracted data into easily readable text by removing unnecessary HTML tags, non-significant words in the report.

Textual Analysis includes all the necessary steps taken to analyse the sentimental score of the text, whether it is positive, negative, or neutral, only on 3 different sections i.e. Management’s Discussion and Analysis (MDA), Quantitative and Qualitative Disclosures about Market Risk (QQDMR), and Risk Factors (RF), in each report.

**Software Requirements**

1. Python 3.5.x or greater or Python IDE, Windows OS (7 or above)
2. A few 3rd party libraries to be installed and upgrading existing libraries

Skip this part if all the libraries installed and up-to-date.

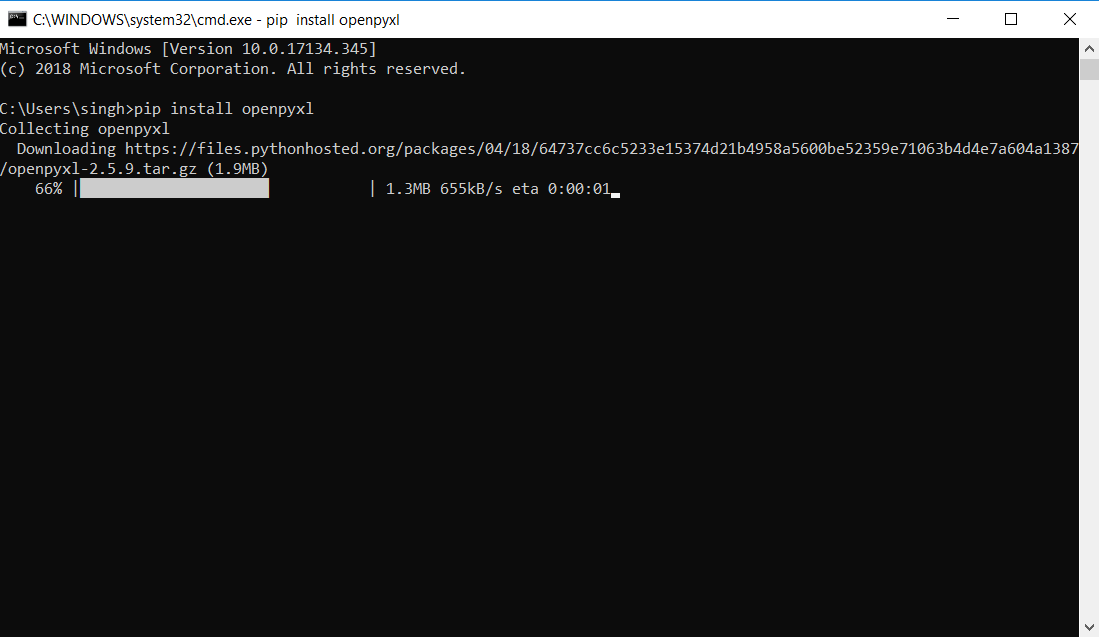
1. Latest version of Python can be downloaded from [www.python.org/downloads](http://www.python.org/downloads) or any good Python IDE can be used. For example: - Anaconda Spyder, PyCharm etc
2. The 3rd party libraries needed are as follows:
3. **openpyxl**

To install this package on Python, open command prompt and type the following:

(Ignore text in RED)

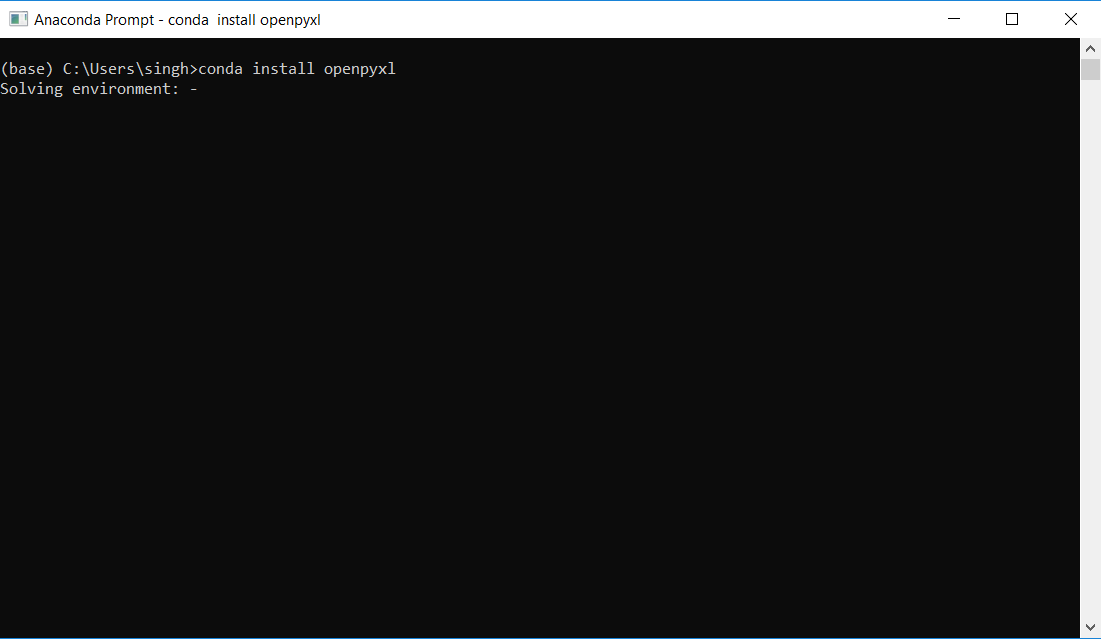
(UserName)> pip install openpyxl

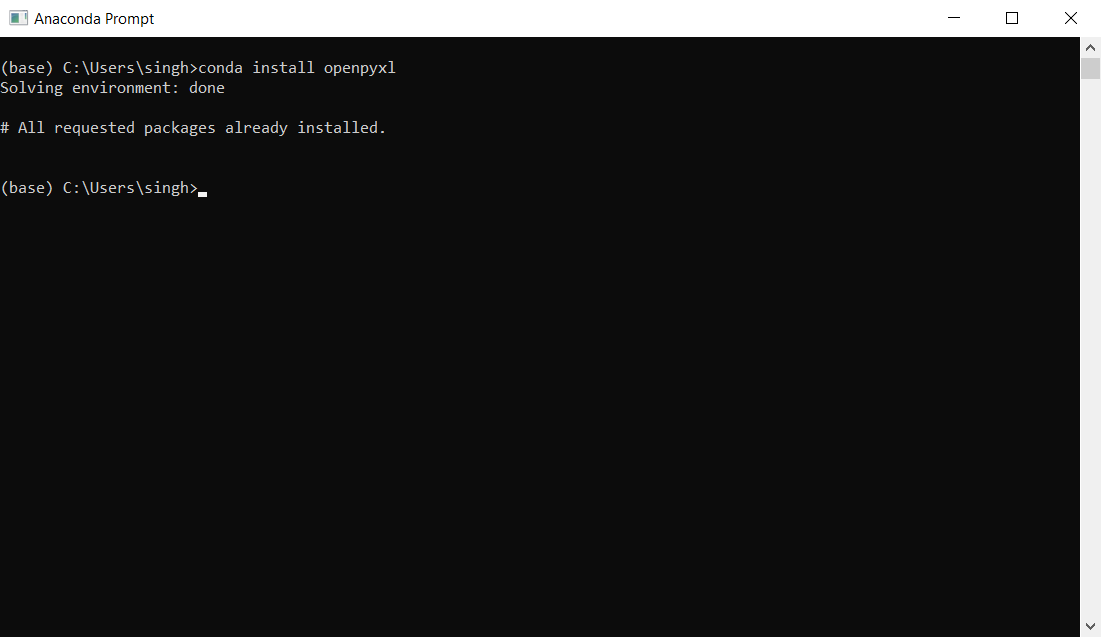
And hit Enter.



To install it in a Python IDE like Anaconda Spyder, open Anaconda Prompt and type the following commands:

(UserName)> conda install openpyxl



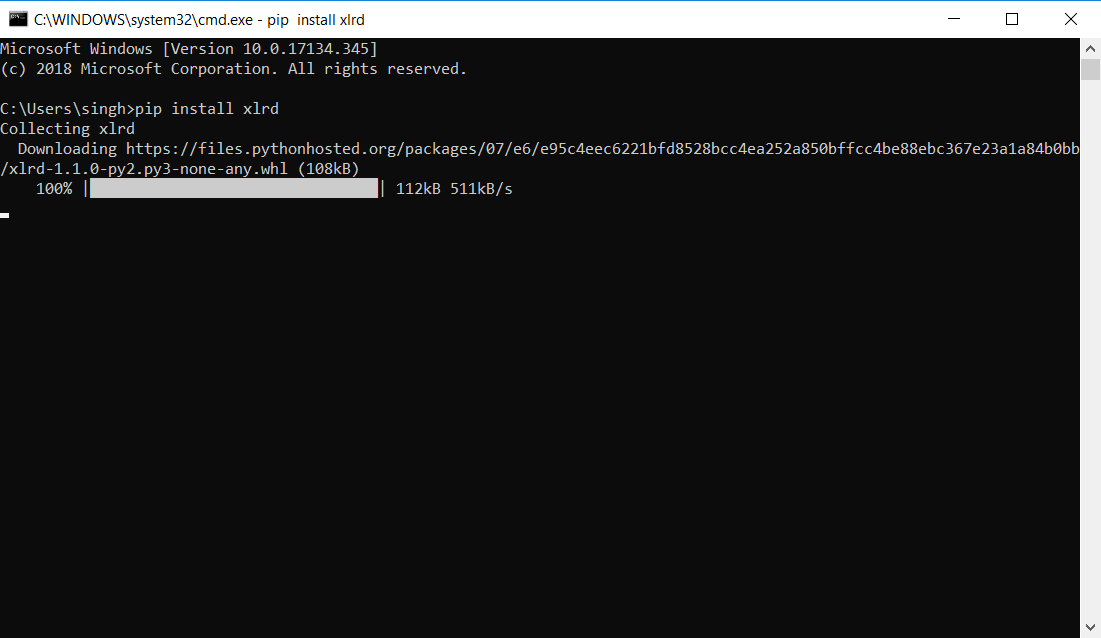


In the above screenshot, “All requested packages already installed” is shown because I have already installed the packages into the system.

1. **xlrd**

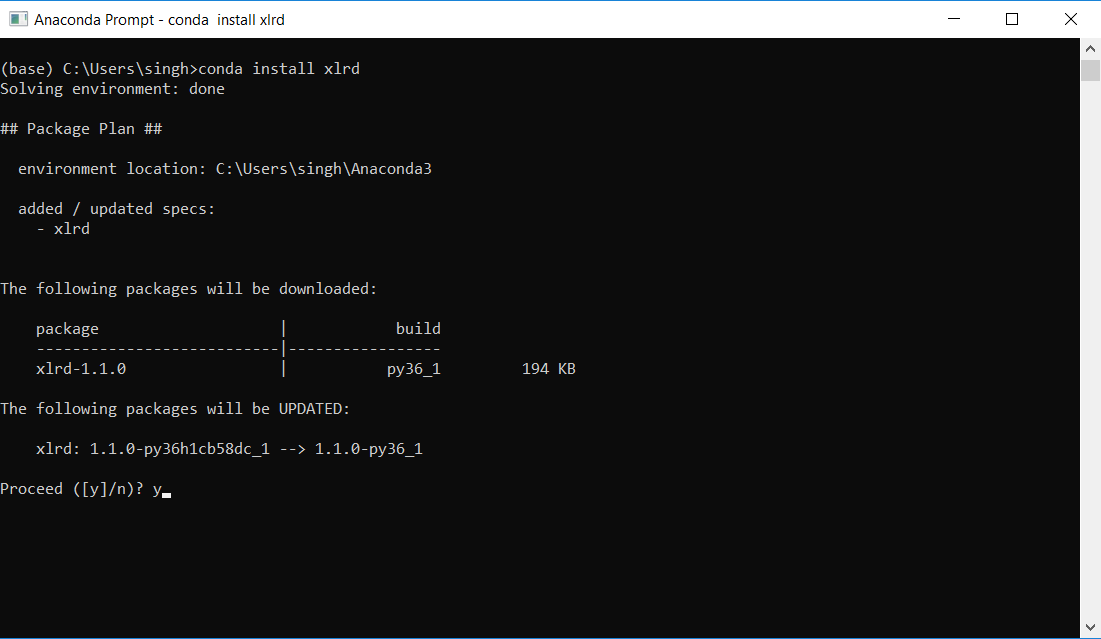
To install this package on Python, type the following commands:

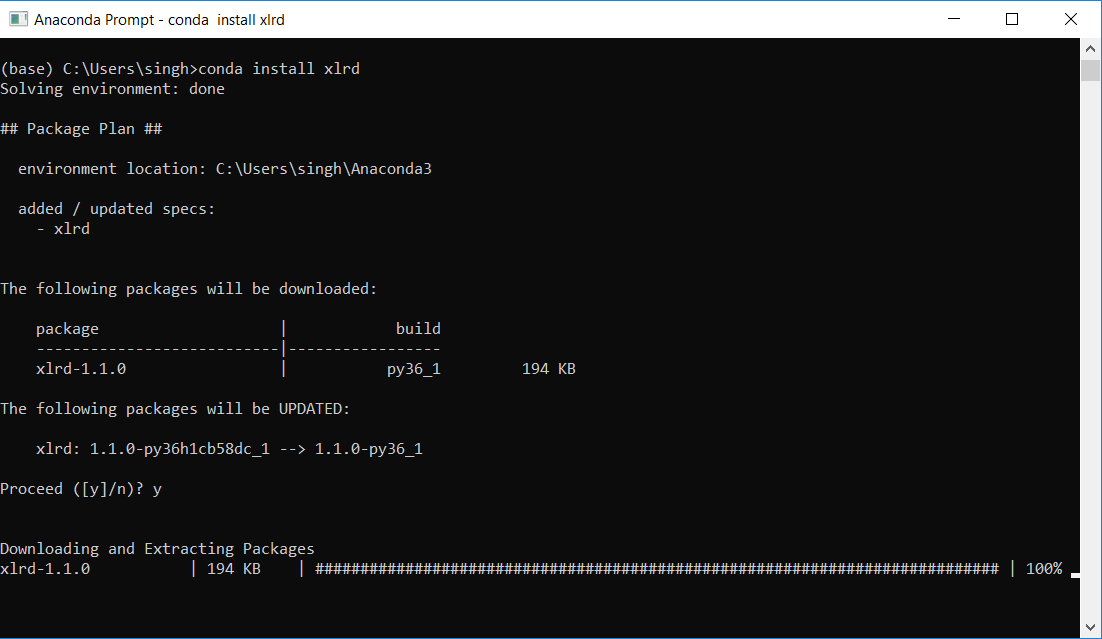
(UserName)> pip install xlrd



To install it in a Python IDE like Anaconda Spyder, open Anaconda Prompt and type the following commands:

(UserName)> conda install xlrd



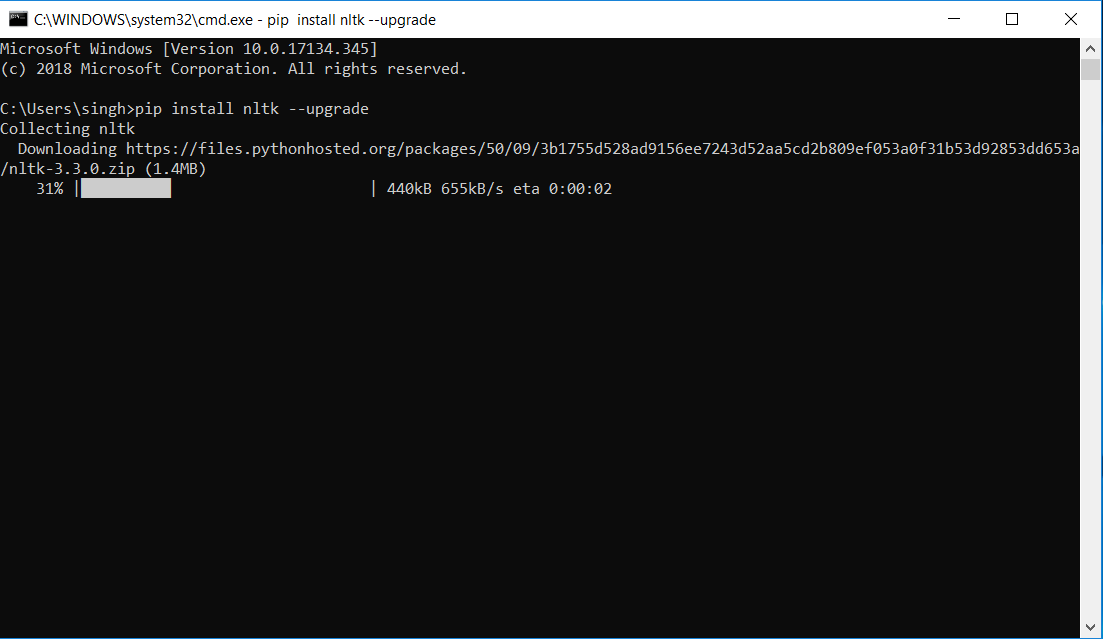


1. **nltk**

As this library is present in the Python packages, we would need to upgrade it to the latest version.

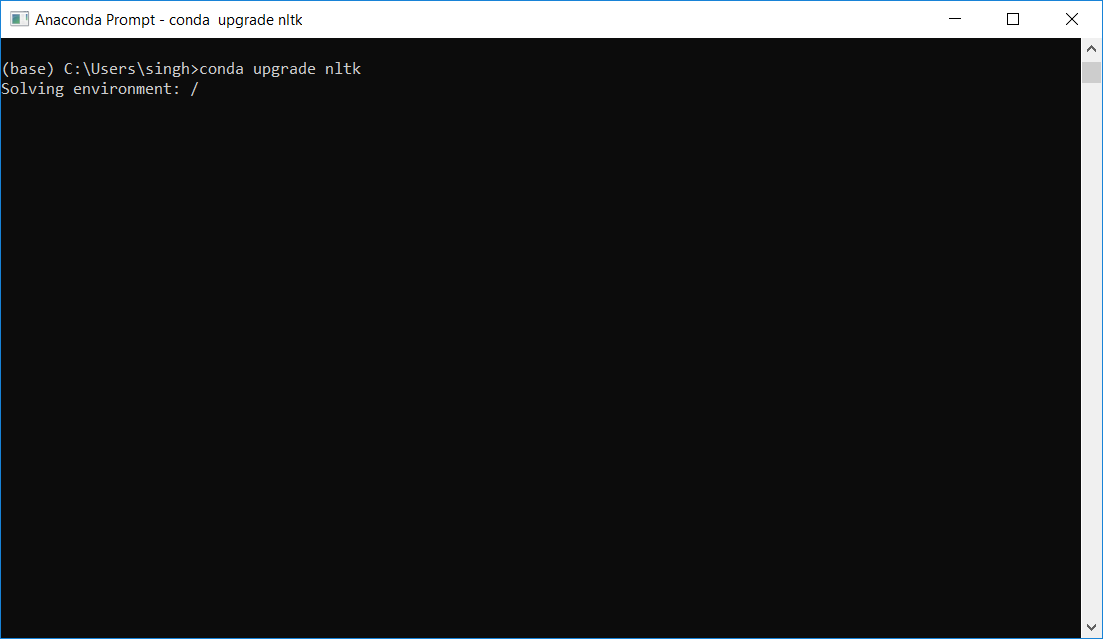
To upgrade it on Python, type the following commands:

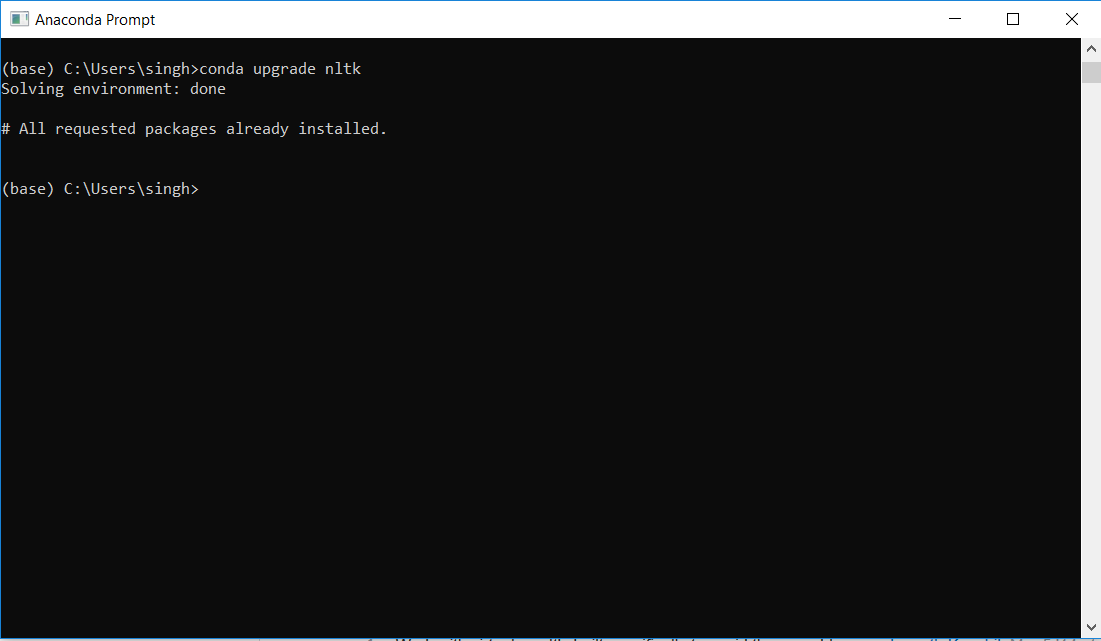
(UserName)> pip install nltk –-upgrade



To upgrade it in a Python IDE like Anaconda Spyder, open Anaconda Prompt and type the following commands:

(UserName)> conda upgrade nltk





In the above screenshot, “All requested packages already installed” is shown because I have already installed the packages into the system.

Now, we are all set to proceed to the next part.

**Modules used**

Different user-defined modules are used in this project, namely “Count.py”, “Scores.py”, “Proportions.py”, “Append\_Data.py”, “complex\_words.py”, “Stopwords.py”, “cw\_c\_whole\_report.py”, “Data\_Extraction.py”, “Data\_Parse.py”, “Fog\_Index.py”, “mda.py”, “qqdmr.py”, “rf.py” and “Main.py”.

Below is a brief explanation of each of the above modules:

* **Count.py**

In this module, there are different functions which are used for text analysis to calculate Word Count, Sentence Count, Average Sentence Length, Constraining Words Count and Uncertainty Words Count.

This module is imported when any of the above functions is used.

* **Scores.py**

In this module, functions to calculate Positive Score, Negative Score and Polarity Score are present.

This module is imported to calculate any of the above three.

* **Proportions.py**

In this module, functions to calculate Positive Words Proportion, Negative Words Proportion, Constraining Words Proportion and Uncertainty Words Proportion are defined.

This module is imported to calculate any of the above four.

* **Append\_Data.py**

This module gets all the data from “cik\_list.xlsx” and “Output Data Structure.xlsx” (provided for format) and copies the data into a new Excel (.xlsx) file named “Output Data Structure.xlsx”

* **complex\_words.py**

This module contains functions which calculate Complex Word Count and Percent of Complex Words in a text.

This module is imported to calculate the above.

* **Stopwords.py**

This module contains functions which store all the stopwords from the text files provided in the Stopwords folder.

* **cw\_c\_whole\_report.py**

This module calculates the number of Constraining Words in the whole report after data parsing and stores the calculated values into “Output Data Structure.xlsx”.

* **Data\_Extraction.py**

This module extracts all the data from each of links given in the “cik\_list.xlsx” file and stores the data into text (.txt) files.

* **Data\_Parse.py**

This module gets all the data stored in text (.txt) files by the above module and parses the HTML tags, aligns the text in order and removes any insignificant text present in it, and stores the resultant in text (.txt) files.

* **Fog\_Index.py**

This module contains a function which calculates Fog Index for a defined text.

* **mda.py**

This module contains a function which does complete analysis of MDA sectioned data and stores the calculated values into “Output Data Structure.xlsx”.

* **qqdmr.py**

This module contains a function which does complete analysis of QQDMR sectioned data and stores the calculated values into “Output Data Structure.xlsx”.

* **rf.py**

This module contains a function which does complete analysis of RF sectioned data and stores the calculated values into “Output Data Structure.xlsx”.

* **Main.py**

This is the Main module. All the above modules work when this module is executed.

(Make sure to execute the code using this module only)

**Usage Example**

Kindly check the Related image **Test.py** file which is created to test the working of text analytical functions used in the actual code.

**Note: -**

It is just a prototype of the actual code used to test the working of the text analytical functions as to get a quick overview of how these functions will work when applied to the actual code.

**Running Code**

Always run the code by executing the Related image **Main.py** file.

Full execution of the code can take time anywhere between 1 – 4 hours, depending on the processing power of the CPU.