**Implementation**

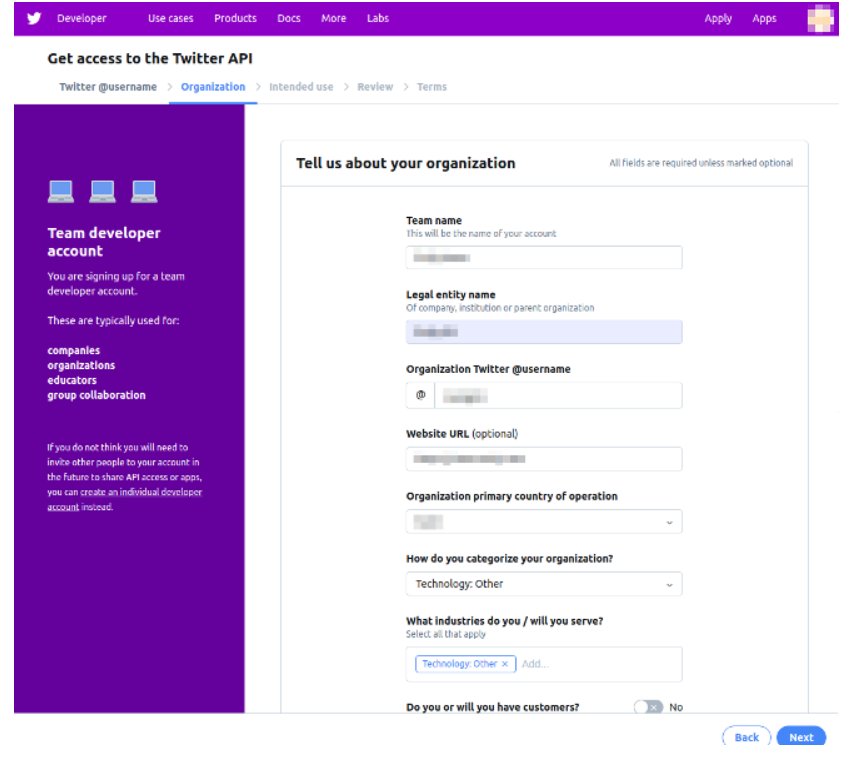
The process is divided in following four subsections:

1. Data Gathering
2. Data Preparing
3. Sentiment analysis model
4. Results
5. Data Gathering: Data gathering is process to collect the data from twitter. This data contains tweet and its meta-data with itself. Meta Data contains data as location of tweet sent, from which device tweet is done, images, links, data and time and etc.
6. Data Preparing: Since the data gathered from the above process possess the meta data which was not required and so there is need to remove this meta data which is done in this data preparing process. The output of this data preparing process gives a readable tweet without any of useless data and is provided to sentiment analysis model.
7. Sentiment Analysis Model: The sentiment analysis is done by pattern analyser model. There are also models which can do classification such as naïve Bayes etc.
8. Results: Checking the polarity and drawing pie chart. Also checking the efficiency of our model.

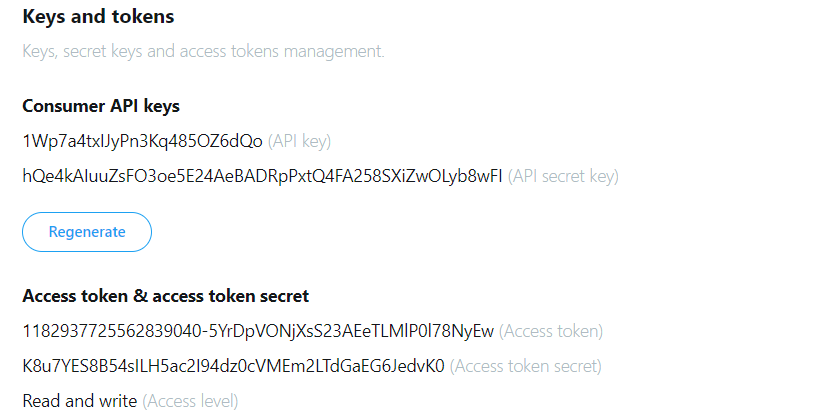
* **Data Gathering**

To apply sentiment analysis, it is required to have data. The data is being retrieved from the twitter database itself, since the data should be of twitter. To retrieve data from twitter there is requirement of creating a developer account. This developer account has permission to retrieve data.

**Creating a developer account**

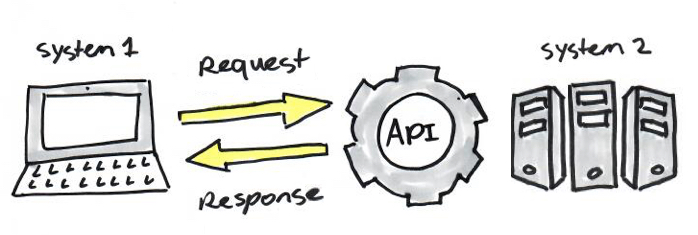
****

**Creating a Twitter developer account**

****

**All required keys**

Above are the keys and tokens for developer to able to retrieve data.



**Basic API work**

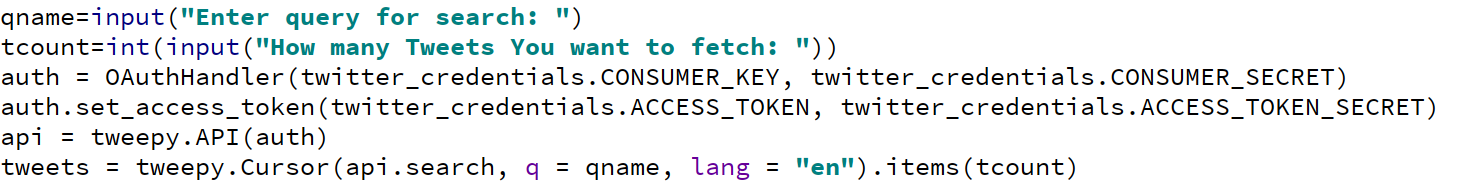
In python there is a library which provides an application package interface with this developer account named as “Tweepy”. This tweepy consists of following classes as:

* API= Twitter API
* OAuthHandler = Authentication Handler
* Cursor = Pagination helper class (used to locate data in correct order)
* Stream = Provide access to streams
* StreamListener = Create our customized Stream

**Fetching Tweets**



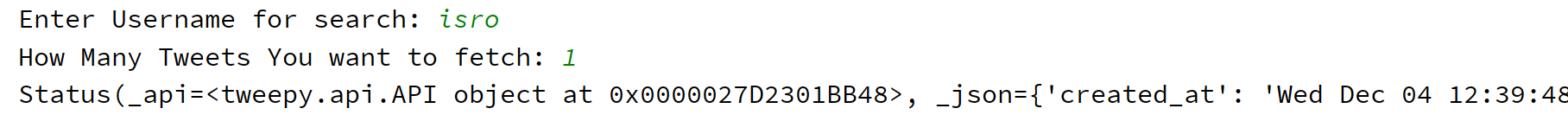
**Importing tweepy library**



**Using OAuth class**

OAuthHandler class requires consumer keys and consumer secret key

OAuthHandler class have a function called set\_access\_token which requires access token and access token secret key



**Getting 1 tweet containing isro as keyword**

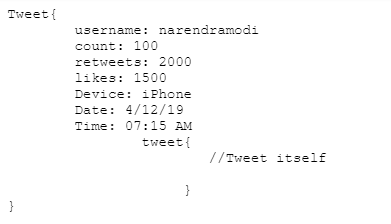


**One tweet with its metadata with tweet highlighted**

Above picture shows one tweet retrieved using tweepy. The tweet contains meta data as shown above. Meta Data contains data as location of tweet sent, from which device tweet is done, images, links, data and time and etc.

**Data Preparing**

The retrieved tweet is a json object by observing the similarities of the json object we can eliminate the meta data and so the cleaned data (or cleaned tweet) is useful for us. A standard tweet json object with less meta data can be viewed as:



**A basic JSON tweet Object**

As we are retrieving only tweet, username, so cleaning count, retweets, likes, time and date and is already been cleaned.

The remaining is kind of string type and so cleaning strings using python is performed as shown in below:

* Cleaning ‘**@**’
* Cleaning ‘**#**’
* Cleaning **links**
* **Clean\_links:** This function takes a string argument (which may or may no contain link like http---) and removes all the links.

def clean\_links(string):

ss = ' '

words = []

ar = string.split(' ') #array of words

for x in ar: #reading one word at time

if not (x.find('http')):

pass

else:

words.append(x)

string = ss.join(words)

print(string)



**String with and without link**

Here a string “Hello my name is Mohit <https://www.google.com/>” contains a link and the clean\_link function will clean the link and output is shown as above.

**Clean\_at\_the\_rate:** This function takes a string argument (which may or may no contain ‘@’ ) and removes all the ‘at the rates’.

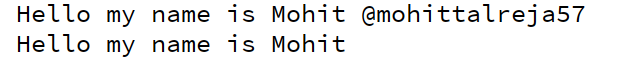
def clean\_at\_the\_rate(word):

for x in word:

if (x == '@'):

return False

return True



**String with and without ‘@’**

**Clean\_hashtags:** This function takes a string argument (which may or may no contain ‘#’ ) and removes all the ‘hashtags ‘#‘’.

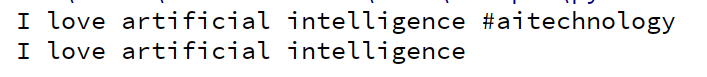
def clean\_at\_the\_rate(word):

for x in word:

if (x == '#'):

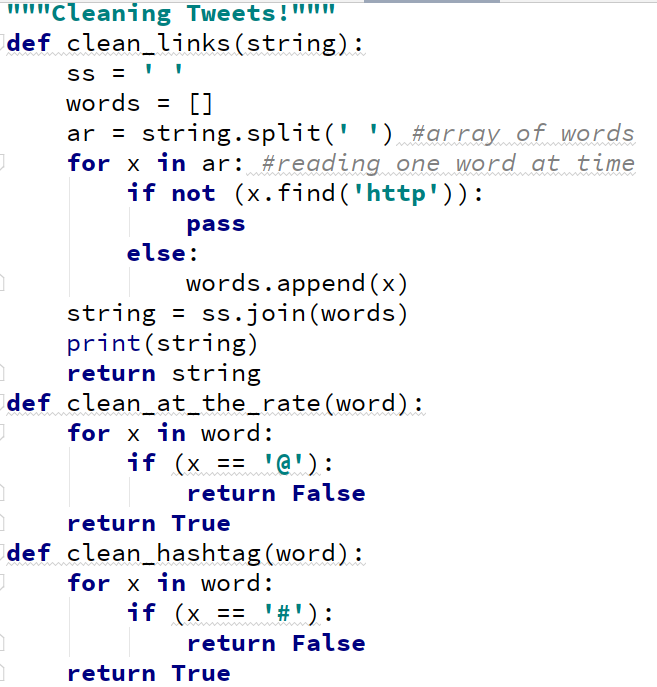
return False

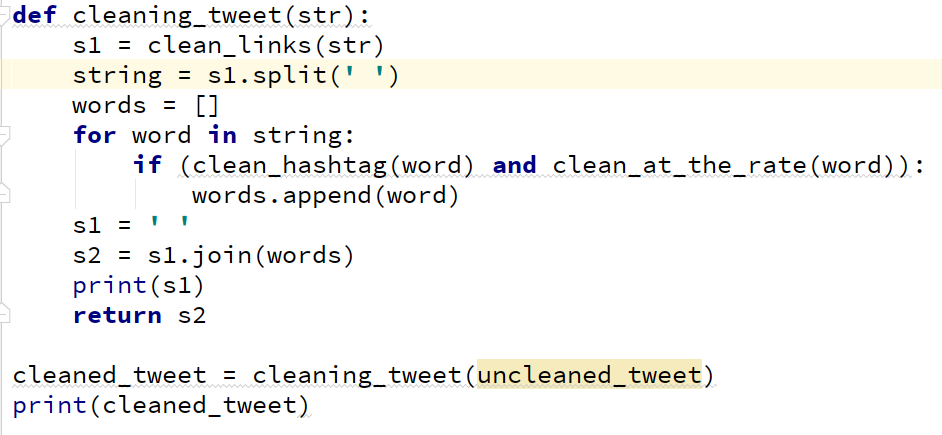
return True



**String with and without ‘#’**

Combining all above functions and giving an example:





Input: I love artificial intelligence #artificialintelligence <https://www.investopedia.com/terms/a/artifcial-intelligence-ai.asp> @mohittalreja

Output: I love artificial intelligence

**Creating a csv file:**

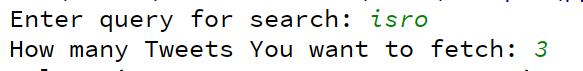
Using pandas library in python the tweet with metadata and without meta data saved in csv file

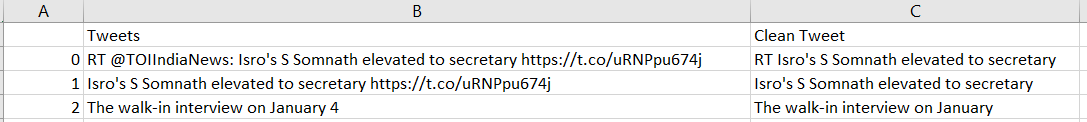
Import pandas as pd

df = pd.DataFrame(data=[tweet.text for tweet in tweetss], columns=['Tweets'])

df['Clean Tweet'] = [cleaning\_tweet(tweet.text) for tweet in tweetss]

Running example:





Since all the meta data, and symbols contained in tweets are cleaned and so now this cleaned tweet is ready for sentiment analysis.

These cleaned tweets are sent to sentiment analysis model which provides the polarity of individual tweets.

**Sentiment analysis model:**

There several types of models used for sentiment analysis. These models differ in some ways as

1. Using Machine Learning approach
2. Using Rule-based approach
3. Machine Learning approach:

Automatic methods, contrary to rule-based systems, don’t rely on manually crafted rules, but on machine learning techniques

The sentiment analysis task is usually modelled as a classification problem where a classifier is fed with a text and returns the corresponding category, e.g. positive, negative.

1. Rules Based approach:

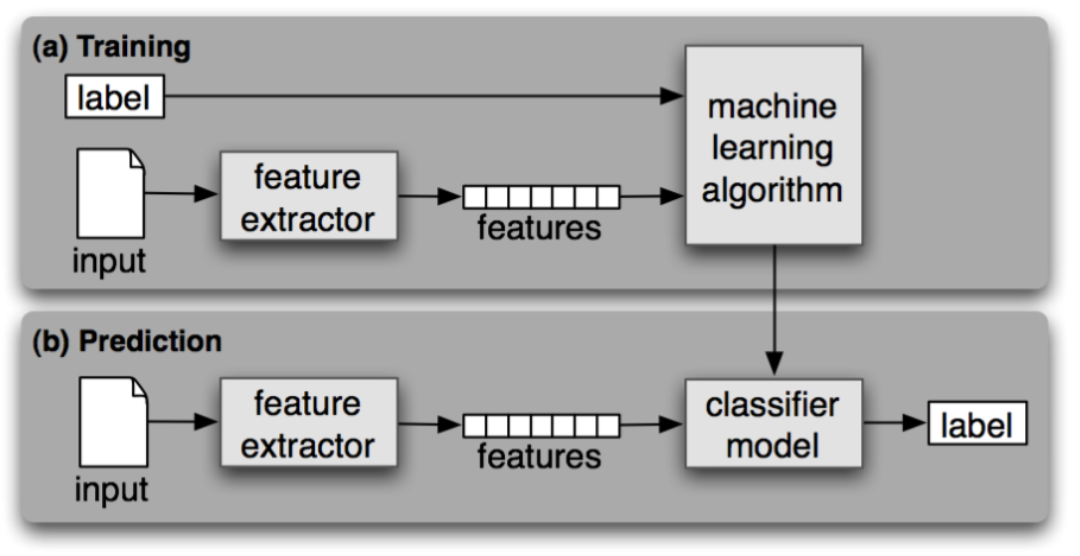
Usually, rule-based approaches define a set of rules in some kind of scripting language that identify subjectivity, polarity, or the subject of an opinion.

For this approach input required can be:

* Classic NLP techniques like stemming, tokenization, part of speech tagging and parsing.
* Other resources, such as lexicons (i.e. lists of words and expressions).

1. **Machine Learning Approach:** Consists of two processes following two processes:

* **The Training Processes**

****

In the training process (a), our model learns to associate a particular input (i.e. a text) to the corresponding output (tag) based on the test samples used for training. The feature extractor transfers the text input into a feature vector. Pairs of feature vectors and tags (e.g. positive, negative, or neutral) are fed into the machine learning algorithm to generate a model.

In the prediction process (b), the feature extractor is used to transform unseen text inputs into feature vectors. These feature vectors are then fed into the model, which generates predicted tags (again, positive, negative, or neutral).

**Feature Extraction from Text**

The first step in a machine learning text classifier is to transform the text into a numerical representation, usually a vector. Usually, each component of the vector represents the frequency of a word or expression in a predefined dictionary (e.g. a lexicon of polarized words). This process is known as feature extraction or text vectorization and the classical approach has been bag-of-words or bag-of-ngrams with their frequency.

**Classification Algorithms:**

The classification step usually involves a statistical model like Naïve Bayes, Logistic Regression, Support Vector Machines, or Neural Networks:

**Naïve Bayes:** A family of probabilistic algorithms that uses Bayes’ Theorem to predict the category of a text.

**Linear Regression:** A very well-known algorithm in statistics used to predict some value (Y) given a set of features (X).

**Support Vector Machines:** A non-probabilistic model which uses a representation of text examples as points in a multidimensional space. These examples are mapped so that the examples of the different categories (sentiments) belong to distinct regions of that space. Then, new texts are mapped onto that same space and predicted to belong to a category based on which region they fall into.

**Deep Learning:** A diverse set of algorithms that attempts to imitate how the human brain works by employing artificial neural networks to process data.

**Note:**

Anyone of the above algorithm can be used to train machine learning model. But since the model created automatically and so it consists errors and so have less accuracy.

Computing error is very important to knowing when your model is “good”, and when it is getting better or worse.

1. **Rule Based approach**

A basic example of a rule-based implementation would be the following:

1. Define three lists of polarized words
   1. negative words such as bad, worst, ugly, etc
   2. positive words such as good, best, beautiful, etc.
   3. neutral words invent, fence, employ, coach, laundry etc.
2. Given a text. Compare each word with database and get their polarity and subjectivity values.
3. Count the number of positive words that appear in the text.
4. Count the number of negative words that appear in the text.
5. If the number of positive word appearances is greater than the number of negative word appearances return a positive sentiment, conversely, return a negative sentiment. Otherwise, return neutral.

A database is created in which each word and their polarities and subjectivities are stored and this database is used.

A snap of the database used



**Text Blob:** TextBlob is a Python library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

Features

* Noun phrase extraction
* Part-of-speech tagging
* Sentiment analysis
* Classification (Naive Bayes, Decision Tree)
* Language translation and detection powered by Google Translate
* Tokenization (splitting text into words and sentences)
* Word and phrase frequencies
* Parsing
* **n**-grams
* Word inflection (pluralization and singularization) and lemmatization
* Spelling correction
* Add new models or languages through extensions
* WordNet integration

First the string is converted to Textblob object. Text blob objects contains above features. This is done as

import textblob

string = “Enter your string here”

blob = TextBlob(string)

Above code have a TextBlob object called blob which contains above mentioned above. From these features polarity is required.

**Using Textblob:**

* The polarity given in output is based on lexicon dictionary made by database.

1. **Polarity of word “Great”**

**Input**:

string = "Great"

blob = TextBlob(string)

print("\n\npolarity of word {} is {}".format(string, blob.polarity))

**Output**:

polarity of word Great is 0.8

1. **Polarity of word “Not Great”**

**Input:**

string = "Not Great"

blob = TextBlob(string)

print("\n\npolarity of word {} is {}".format(string, blob.polarity))

**Output:**

polarity of word Not Great is -0.4

1. **Polarity of word “Very”**

**Input:**

string = "Very"

blob = TextBlob(string)

print("\n\npolarity of word {} is {}".format(string, blob.polarity))

**Output:**

polarity of word Very is 0.2

1. **Adding negation ‘not‘to word or sentence**

**Input:**

string = "Not great"

blob = TextBlob(string)

print("\n\npolarity of {} is {}".format(string, blob.polarity))

**Output:**

polarity of Not great is -0.4

1. **Adding adjective like ‘Very’ to word or sentence**

**Input:**

string = "Very great"

blob = TextBlob(string)

print("\n\npolarity of {} is {}".format(string, blob.polarity))

**Output:**

polarity of very great is 1.0

1. **Addition of negation and adjective**

**Input:**

string = "not very great"

blob = TextBlob(string)

print("\n\npolarity of {} is {}".format(string, blob.polarity))

**Output:**

polarity of not very great is -0.30769

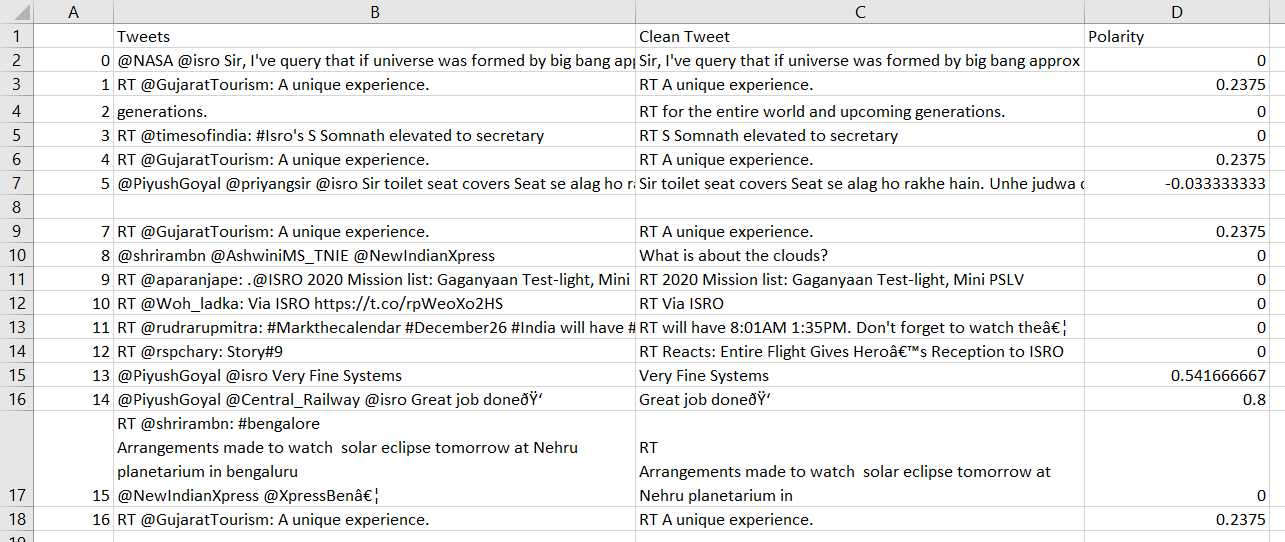
Above result are obtained as

Polarity = (polarity of ‘not’)\*(polarity of ‘great’)

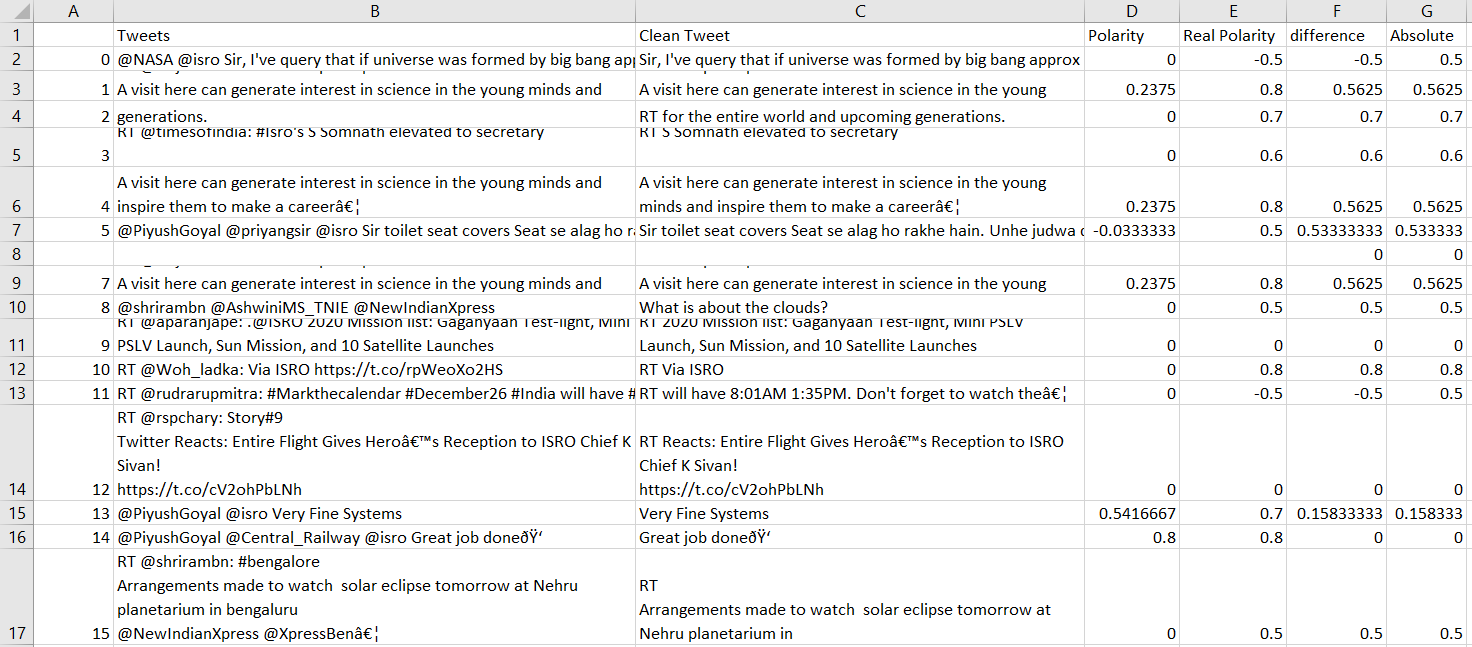
------------------------------------------------------------

(intensity of ‘very’)

**Results:**



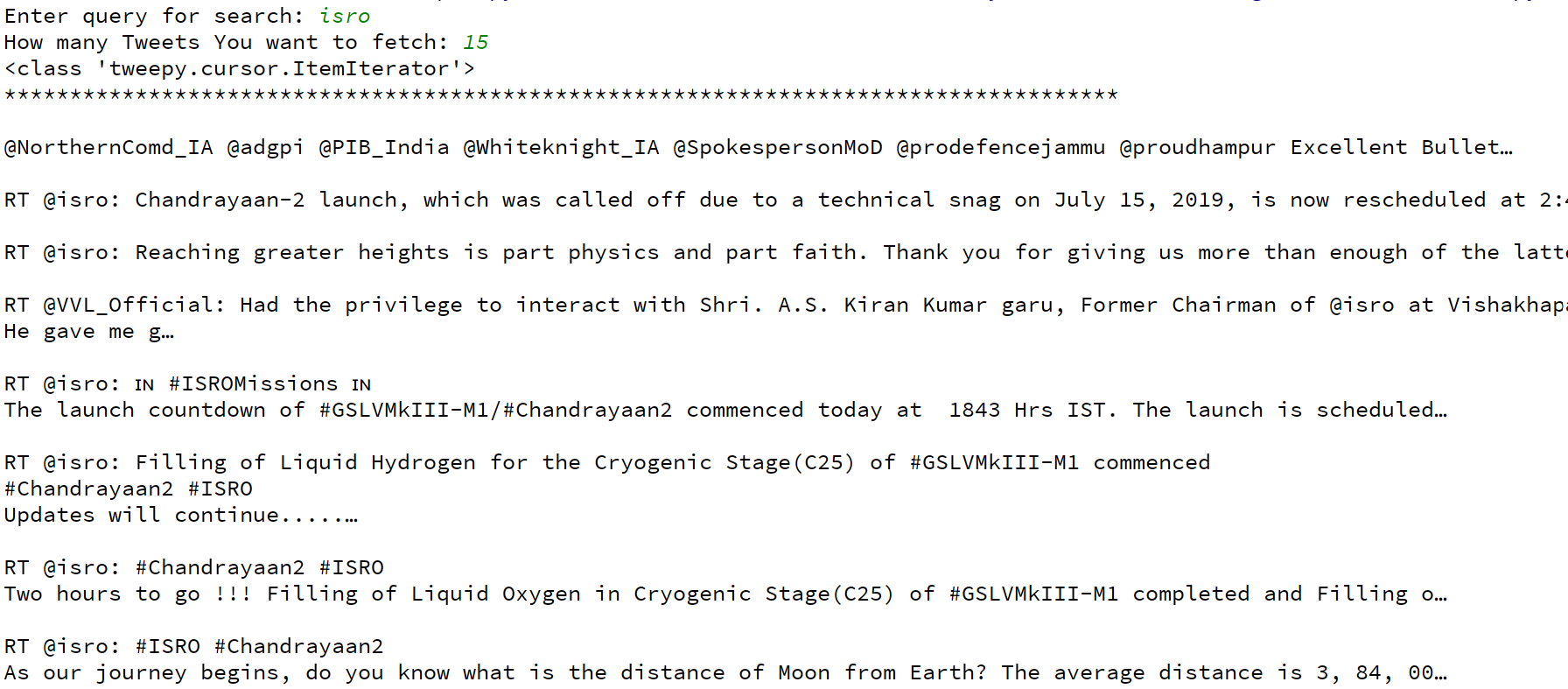
**Tweet, cleaned tweet and their polarity**

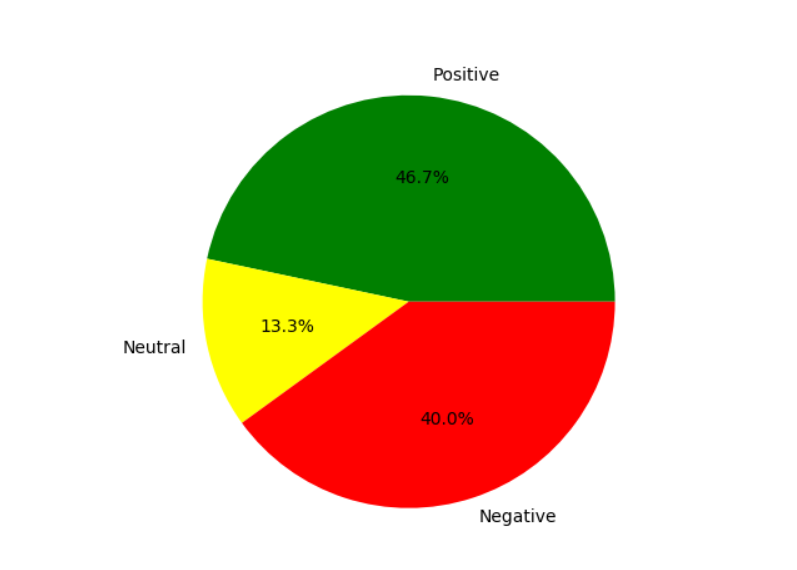
****

**Tweet, cleaned tweet and their polarity and real polarity**

The real polarity is obtained from already made NLTK sentiment analysis model.

**Example 1:** Getting 15 tweets of ISRO

****

**Drawing Pie chart**

**ISRO 15 tweets**

**References:**

* textblob library functions

https://textblob.readthedocs.io/en/dev/quickstart.html#create-a-textblob

* database of lexicon and their polarity:

https://planspace.org/20150607-textblob\_sentiment/

* methodology of sentiment analysis

https://towardsdatascience.com/deep-learning-for-sentiment-analysis-7da8006bf6c1

* types of sentiment analysis models

<https://medium.com/@datamonsters/sentiment-analysis-tools-overview-part-2->7f3a75c262a3

* NLTK premade model for testing our mode

https://text-processing.com/demo/sentiment/