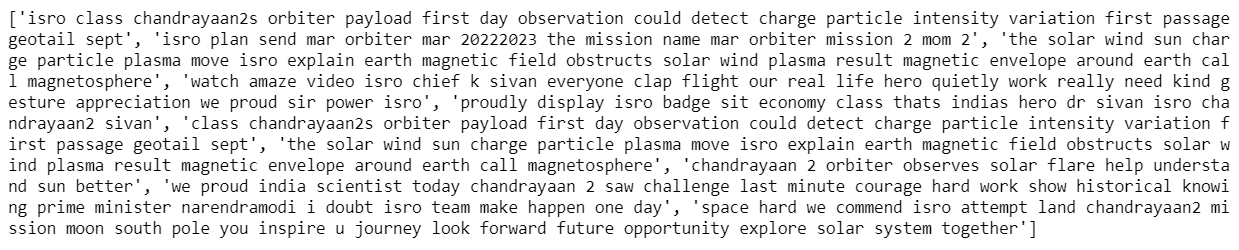
**Q1. Find 10 short text-items (20-30 words); they could be emails, short docs, tweets or whatever. Make sure they all deal with some common topic of interest; so they have some of the same words.**

1. **Remove the standard stopwords from them using some standard list, use nltk.**

**Ans**: Following steps are followed to pre-process the data.

* **TweetTokenizer(Tokenize the tweets) -> Stop word removal -> Removal of special characters -> Changing words to lower case -> Blank space removal -> Applying WordNetLemmatizer -> Joined elements of multiple list to a single List**
* After applying the above steps on tweets below is the produced clean corpus.



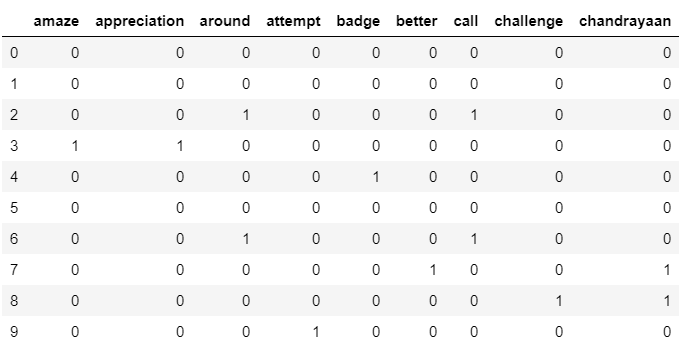
**Fig: Preprocessed Tweets**

1. **Compute the TF scores for all the remaining words in the texts and use R to show the word-cloud for these words. In your answer provide the matix of TF scores and the word-cloud image.**

**Ans:Term frequency:** Measures how frequently a term occurs in a document.

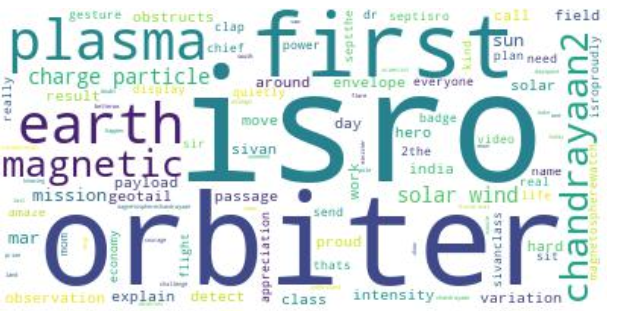
**TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)**

* To create a word-cloud in python we need **CountVectorizer** inside the package called as **sklearn.feature\_extraction.text**.
* Created an object of CountVectorizer as **cnt\_vec**.
* Performed **cnt\_vec.fit\_transform(corpus)**.The fit\_transform method calculates the frequency of each word in the entire sets of docs.
* Now to represent calculated frequency in a table we use Dataframe. Below is the snapshot of the frequency table.



**Fig:Snapshot of Frequency table produced after applying TF-score**

* Below is the word cloud that is produced based on the tweets corpus.



**Fig: Word Cloud based on Tweets**

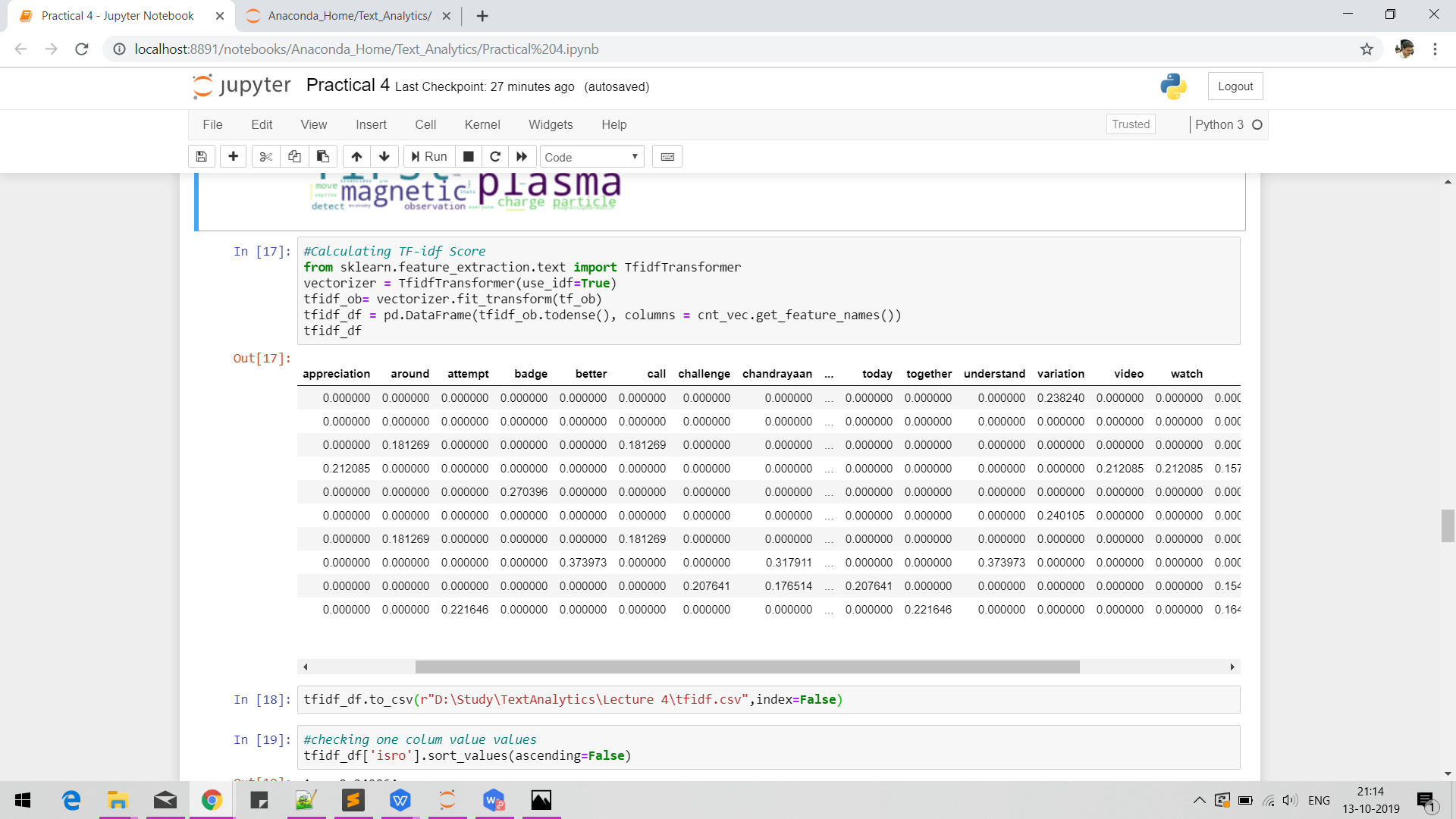
1. **Now, compute the TF-IDF scores for all the same words in the texts. Construct a set of words that represents the TF-IDF scores you have found, for all the words. Use R to show a word-cloud for these words. Also, provide the matrix of TF-IDF scores and the word-cloud image.**

**Ans: Inverse document frequency(IDF):** The specificity of a term can be quantified as an inverse function of the number of documents in which it occurs. Actually it measures how important a term is.

The Formula is defined as: **IDF(t) = log\_e(Total number of documents / Number of documents with term t in it).**

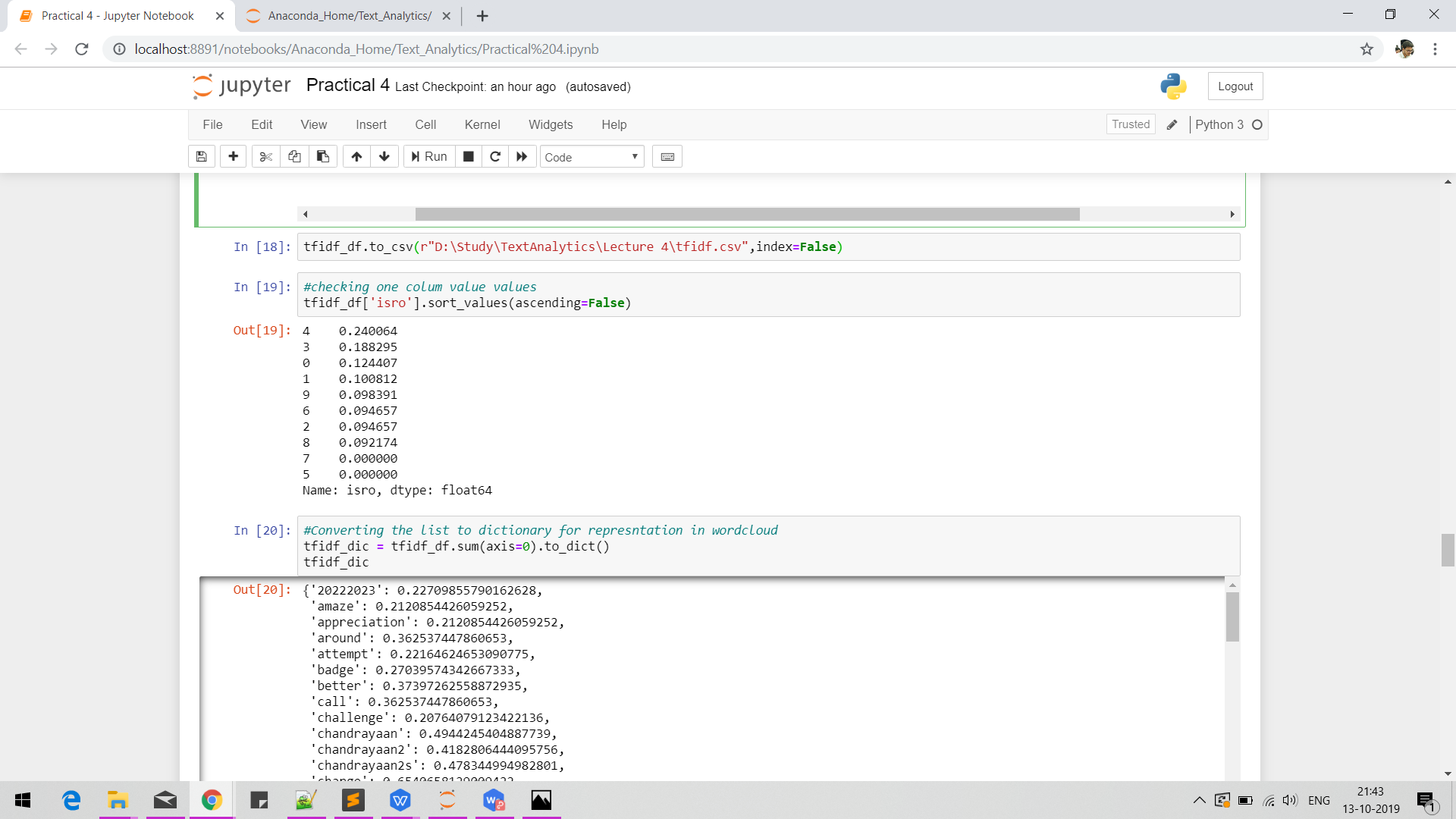
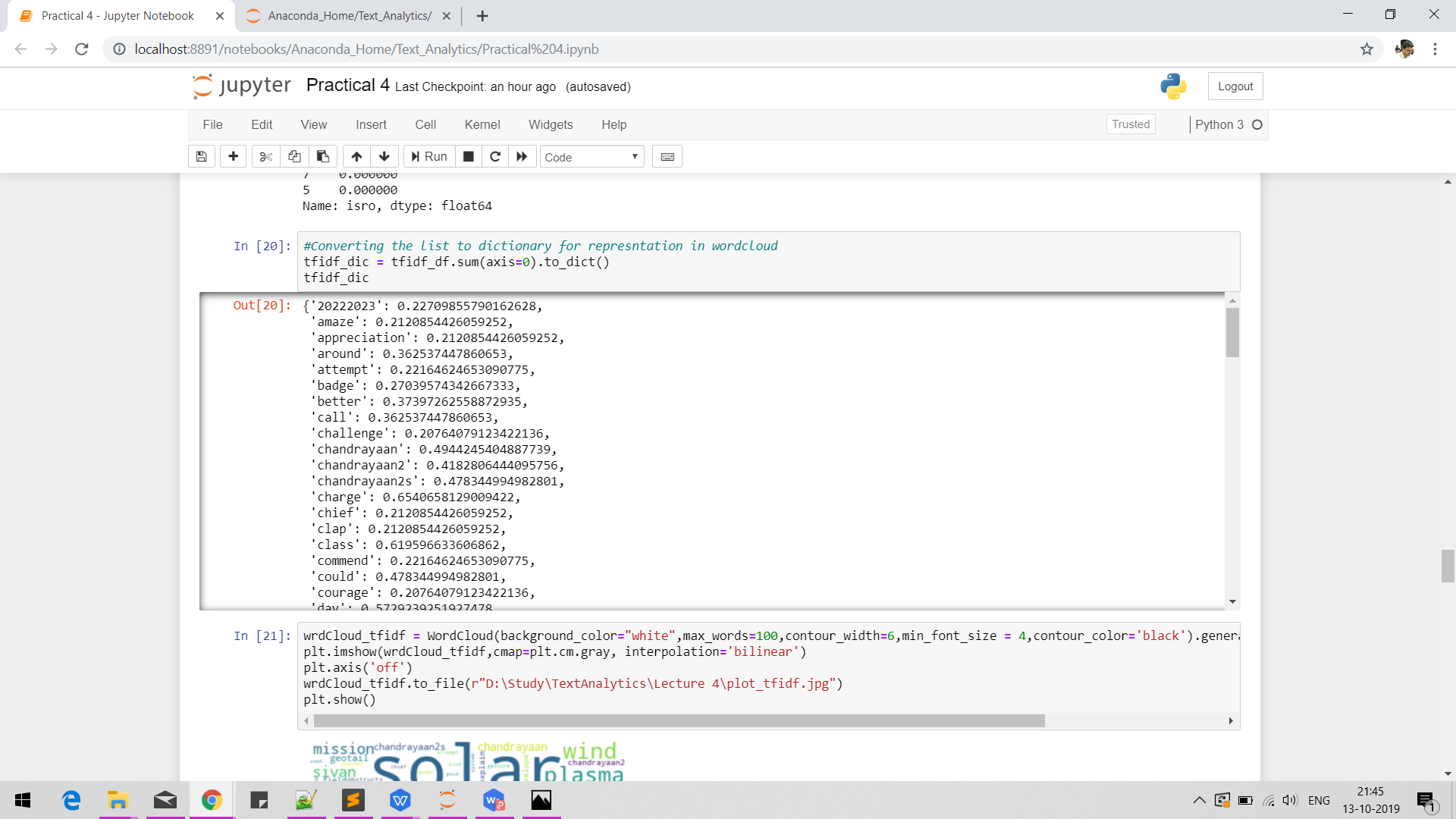
**TF-IDF** is calculated using

* To calculate tf-idf in python we need a module **TfidfTransformer** inside package **sklearn.feature\_extraction.text**
* fit\_transform() method of TfidfTransformer is used to calculate the TF-score from pre-processed data.
* In the matrix below, left side shows the number of documents and the top column shows the words occuring in each document.

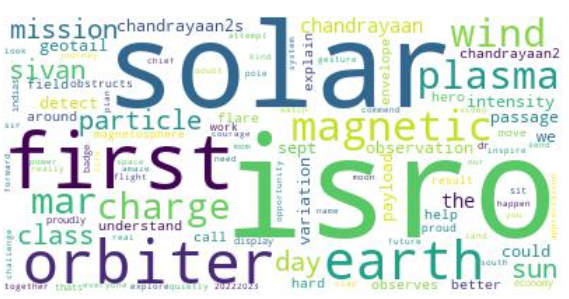


**Fig: TF-IDF Matrix**

* Before creating the word cloud, summation of tf-idf of each columns i.e words are calculated and a dictionary is created. Below is the snippet of the dictionary and tf-idf summation .

* Below is the word-cloud out of all words



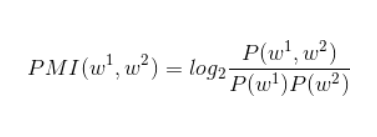
**Fig: Word cloud based on TF\_IDF-Score**

**Q2. Using Python or R, compute the PMI scores for all adjacent pairs of words in your 10-doc corpus (ie the texts after stop-word removal).**

**List the top-10 pairs based on the PMI scores found for the pairs.**

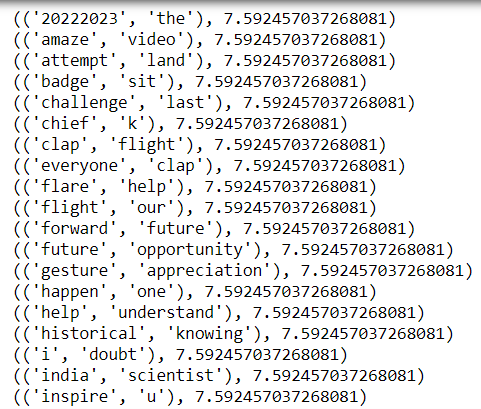
**Do the results make sense? If not, then introduce a minimal cut-off frequency and re-compute the top-10 until they seem sensible.**

**Ans:** Pointwise Mutual Information (PMI) : The main intuition is that it measures how much more likely the words co-occur than if they were independent. formula is given by:

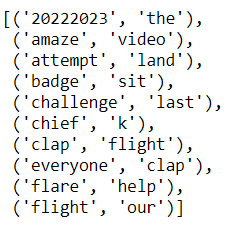


* Module required to calculate the PMI in python is **nltk.collocations.BigramAssocMeasures()**
* Corpus of 10 documents id tokenized and bigrams are found using BigramCollocationFinder.
* The score is produced using **bigram\_measures.pmi.**

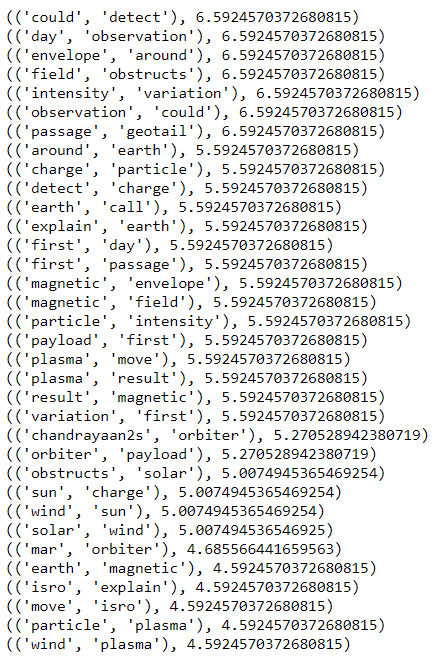
Below is the snippet of bigram with pmi score:



* Top 10 pairs based on PMI score:



* Looking at the result it can be seen that it doesn’t make much sense. The issue with PMI is that it over-estimates the low frequency events because of how it treats counts.
* We can set a frequency filter to narrow down our result. To do so I have used apply\_freq\_filter(). Below is the output after applying frequency filter=2.



**Q3.Entropy has been used to determine whether tweet set is interesting (contains variety) or repetitive**

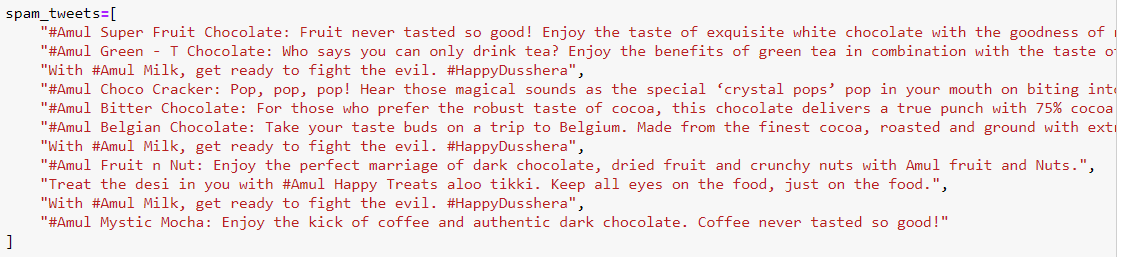
**(spam). Create two sets of 10 made-up tweets:**

1. **spam-set: where the 10 tweets are very similar containing an advert for a product.**
2. **b. random-set: where the 10 tweets are very different, chosen at random from Twitter. Now, find a Python/R program or package that computes entropy and find the entropy values for (i)**

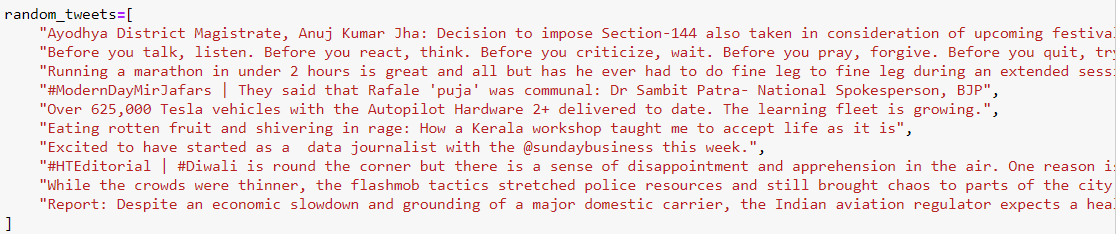
**spam-set, (ii) random-set, (iii) the two sets combined. Report the program you used and its source, the tweet data and the entropy values found.**

**Ans:** Two sets are created,

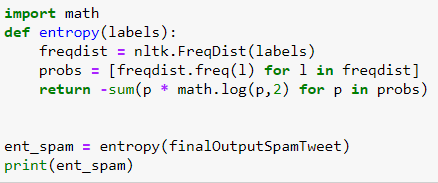
Twitter Spam set:



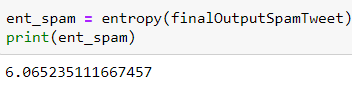
Random Set:



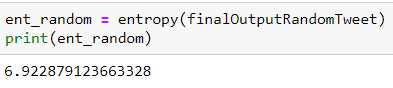
* Entropy is defined as the sum of the probability of each label times the log probability of that same label, written as H(A) = -sum(p \* log(p), axis=0). It is randomness in words.
* FreqDist calculates the frequency distribution for each token and prob generates the probability of each token and entropy is returned(-sum(p\*log(p with base 2))) for every probability [1]
* Below is the function to calculate the entropy.



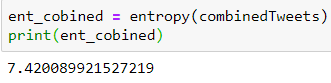
1. Entropy is calculated for the spam set is shown below:



1. Entropy is calculated for the random set is shown below:



1. Combined entropy is shown below:



**References:**

[1] (Natural Language Processing with Python,2019 , Steven Bird, Ewan Klein and Edward Loper)