1. Problem:

Sentiment Analysis Over 3 Feelings with 60.000 Tweets

In the past decade, new forms of communication, such as text messaging have emerged. While there is no limit to the range of information conveyed by tweets and texts, often these short messages are used to share opinions and sentiments that people have about what is going on in the world around them.

Working with these informal text genres presents challenges for natural language processing beyond those typically encountered when working with more traditional text genres, such as newswire data. Tweets and texts are short: a sentence or a headline rather than a document. The language used is very informal, with creative spelling and punctuation, misspellings, slang, new words, URLs, and genre specific terminology and abbreviations, such as, RT for "re-tweet" and #hashtags, which are a type of tagging for Twitter messages. How to handle such challenges so as to automatically mine and understand the opinions and sentiments that people are communicating has only very recently been the subject of research.

I believe that a freely available, annotated corpus that can be used as a common testbed is needed in order to promote research that will lead to a better understanding of how sentiment is conveyed in tweets and texts. My primary goal in this task is to create such a resource: a corpus of tweets and texts with sentiment expressions.

Sentiment Analysis- Emotion in Text in a variation on the popular task of sentiment analysis, this dataset contains labels for the emotional content (such as happiness, sadness, and neutral) of texts. 60.000 samples across 3 labels.

2. Client:

a. Businesses/ Organizations

Sentiment analysis has many applications and benefits to any business and organization. It can be used to give valuable insights to the business regarding how people feel about your product brand or service.

b. **Product Advocates/ Social Media Influencers**

When applied to social media channels, it can be used to identify spikes in sentiment, thereby allowing someone to identify potential product advocates or social media influencers.

It can be used to identify when potential negative threads are emerging online regarding your business, thereby allowing you to be proactive in dealing with it more quickly.

c. **Corporate Network**

Sentiment analysis could also be applied to the corporate network, for example, by applying it to its email server, emails could be monitored for their general "tone". For example, Tone Detector is an Outlook Add-in that determines the "tone" of your email as you type. Like an emotional spell checker for all of your outgoing email.

3. Data set:

a. Taken from : https://data.world/crowdflower/sentiment-analysis-in-text

b. Data Set includes 4 features and 60.000 data points/samples.

c. This data comes from 2016. By means of APIs, we can collect fresh data from 2017 and 2018.

4. Approach:

a. This is a supervised problem. Target Feature will be 'sentiment'. Since this feature has 13 labels, solution would be multi label classification problem, or other approach might be a regression.

b. To be able to pre-process the data, I will use NLP methods on the content.

5. Data Preprocessing:

Sentiment Analysis examines the problem of studying texts, like posts and reviews, uploaded by users on microblogging platforms, forums, and electronic businesses, regarding the opinions they have about a product, service, event, person or idea.

***a. Data Wrangling*:**

(1) Pre-processing

An initial step in text and sentiment classification is pre-processing. A significant amount of techniques is applied to data in order to reduce the noise of text, reduce dimensionality, and assist in the improvement of classification effectiveness. The most popular techniques include:

- ✓ **Removing special characters:** Special characters and symbols which are usually non alphanumeric characters often add to the extra noise in unstructured text. More than often, simple regular expressions (regexes) can be used to achieve this.

- ✓ **Stemming and lemmatization:** Word stems are usually the base form of possible words that can be created by attaching affixes like prefixes and suffixes to the stem to create new words. This is known as inflection. The reverse process of obtaining the base form of a word is known as stemming. A simple example are the words WATCHES, WATCHING, and WATCHED. They have the word root stem WATCH as the base form. Lemmatization is very similar to stemming, where we remove word affixes to get to the base form of a word. However the base form in this case is known as the root word but not the root stem. The difference being that the root word is always a lexicographically correct word (present in the dictionary) but the root stem may not be so.

- ✓ **Removing accented characters:** In any text corpus, especially if you are dealing with the English language, often you might be dealing with accented characters\letters. Hence we need to make sure that these characters are converted and standardized into ASCII characters. A simple example would be converting é to e.

- ✓ **Removing HTML Tags:** Our text often contains unnecessary content like HTML tags, which do not add much value when analyzing text. The BeautifulSoup library does an excellent job in providing necessary functions for this.

- ✓ **Removing punctuation** (removing extra white spacing)

- ✓ **Text Lower Casing**

- ✓ **Removing stopwords:** Words which have little or no significance especially when constructing meaningful features from text are known as stopwords or stop words. These are usually words that end up having the maximum frequency if you do a simple term or word frequency in a corpus. Words like a, an, the, and so on are considered to be stopwords. There is no universal stopword list but we use a standard English language

stopwords list from nltk. You can also add your own domain specific stopwords as needed.

```
dfAdd['NewContent'] = dfAdd['NewContent'].\
apply(lambda x: [wordnet_lemmatizer.lemmatize(a) for a in
                            [k for k in [l for l in [t.lower() for t in word_tokenize(x)]
                                         if l.isalpha()] if k not in english_stops]]).\
apply(lambda x: " ".join(x))
```

(2) Bag of Words (Plus n-grams) (CountVectorizing in ScikitLearn)

Bag of Words is the vector space representational model for unstructured text. It is simply a mathematical model to represent unstructured text (or any other data) as numeric vectors, such that each dimension of the vector is a specific feature/attribute. The bag of words model represents each text document as a numeric vector where each dimension is a specific word from the corpus and the value could be its frequency in the document, occurrence (denoted by 1 or 0) or even weighted values.

zikihekai	zillaman	zillygrl	zimbabwe	zimmer	zimmermann	zinc	zincroof	zindelayentl	zinedistro	zing
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

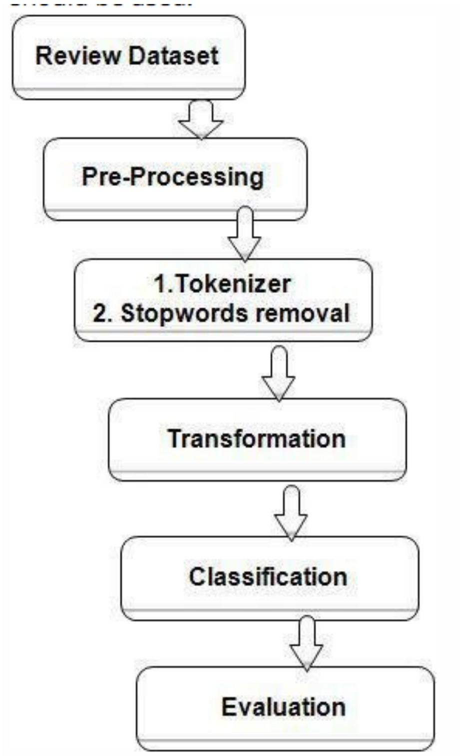
(3) TF-IDF Vectorizer

There are some potential problems which might arise with the Bag of Words model when it is used on large corpora. Since the feature vectors are based on absolute term frequencies, there might be some terms which occur frequently across all documents and these may tend to overshadow other terms in the feature set. The TF-IDF model tries to combat this issue by using a scaling or normalizing factor in its computation. TF-IDF stands for Term Frequency-Inverse Document Frequency, which uses a combination of two metrics in its computation, namely: term frequency (tf) and inverse document frequency (idf). This technique was developed for ranking results for queries in search engines and now it is an indispensable model in the world of information retrieval and NLP.

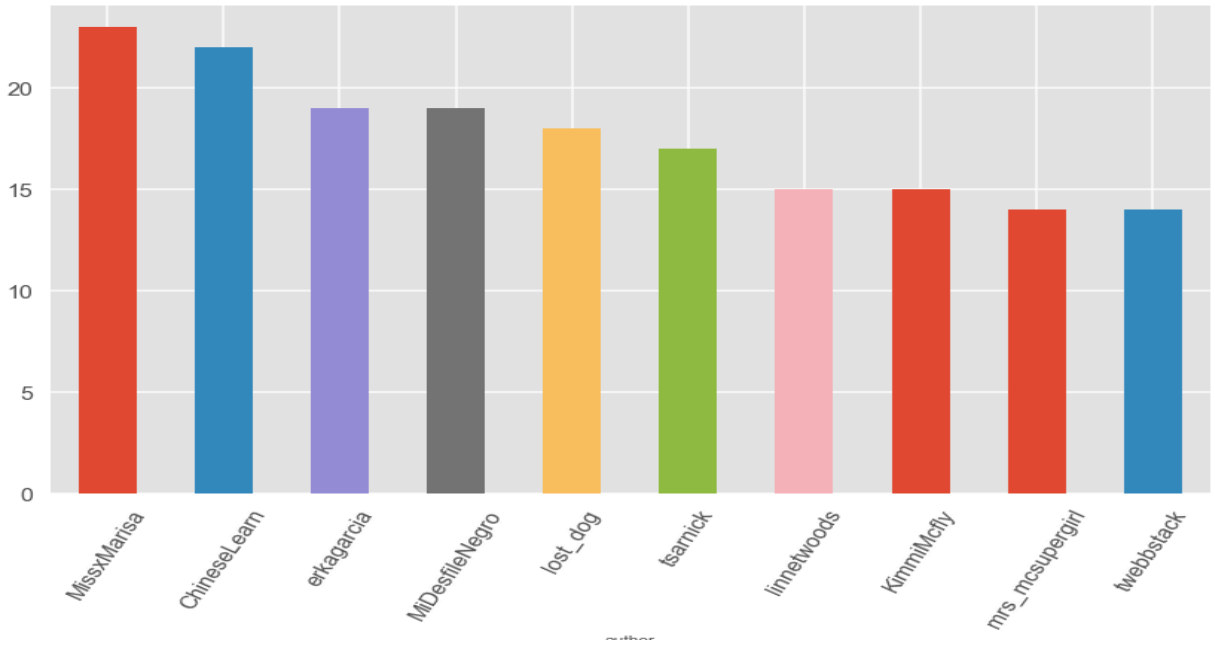
Mathematically, we can define TF-IDF as tfidf = tf x idf, which can be expanded further to be represented as follows.

Here, tfidf(w, D) is the TF-IDF score for word w in document D. The term tf(w, D) represents the term frequency of the word w in document D, which can be obtained from the Bag of Words model. The term idf(w, D) is the inverse document frequency for the term w, which can be computed as the log transform of the total number of documents in the corpus C divided by the document frequency of the word w, which is basically the frequency of documents in the corpus where the word w occurs. There are multiple variants of this model but they all end up giving quite similar results. Let's apply this on our corpus now!

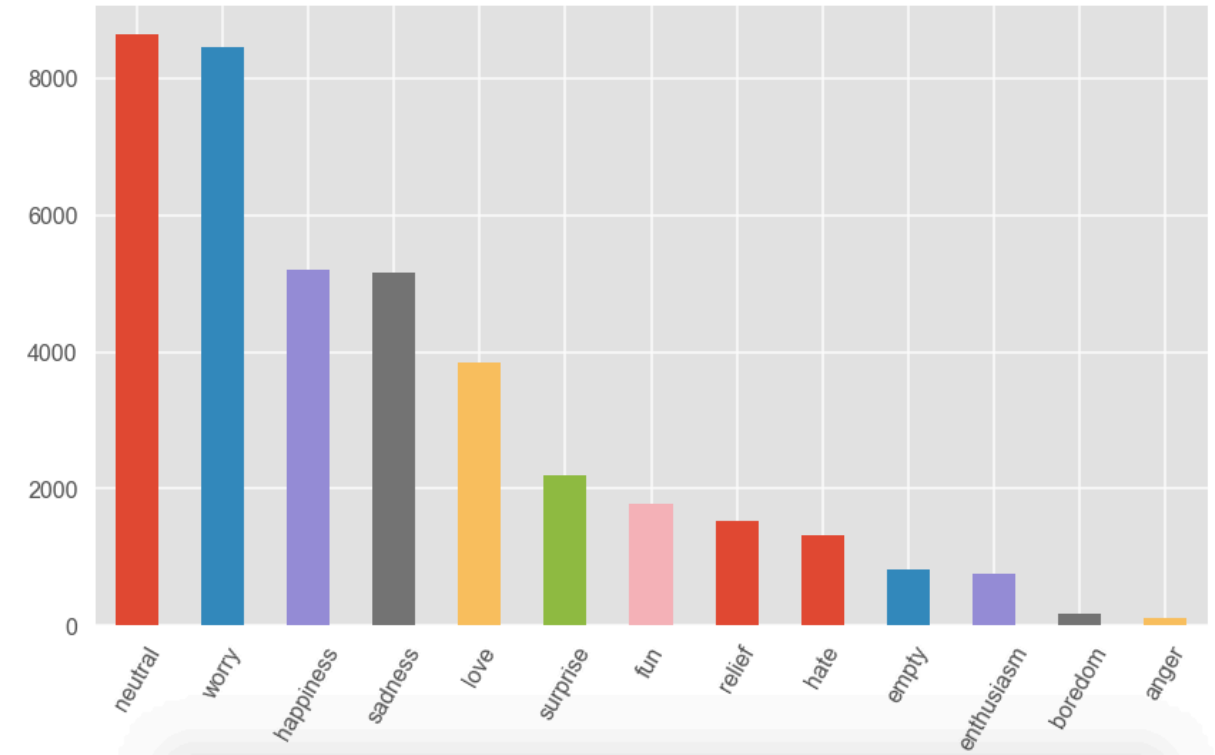
grl	absolves	absoulutely	absoulutley	absoute	abstractg	abstraction	absurd	abt	abueltia	abundantly	abuse	abusing	abusive	abuzz	aby	abzquine	ac
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0



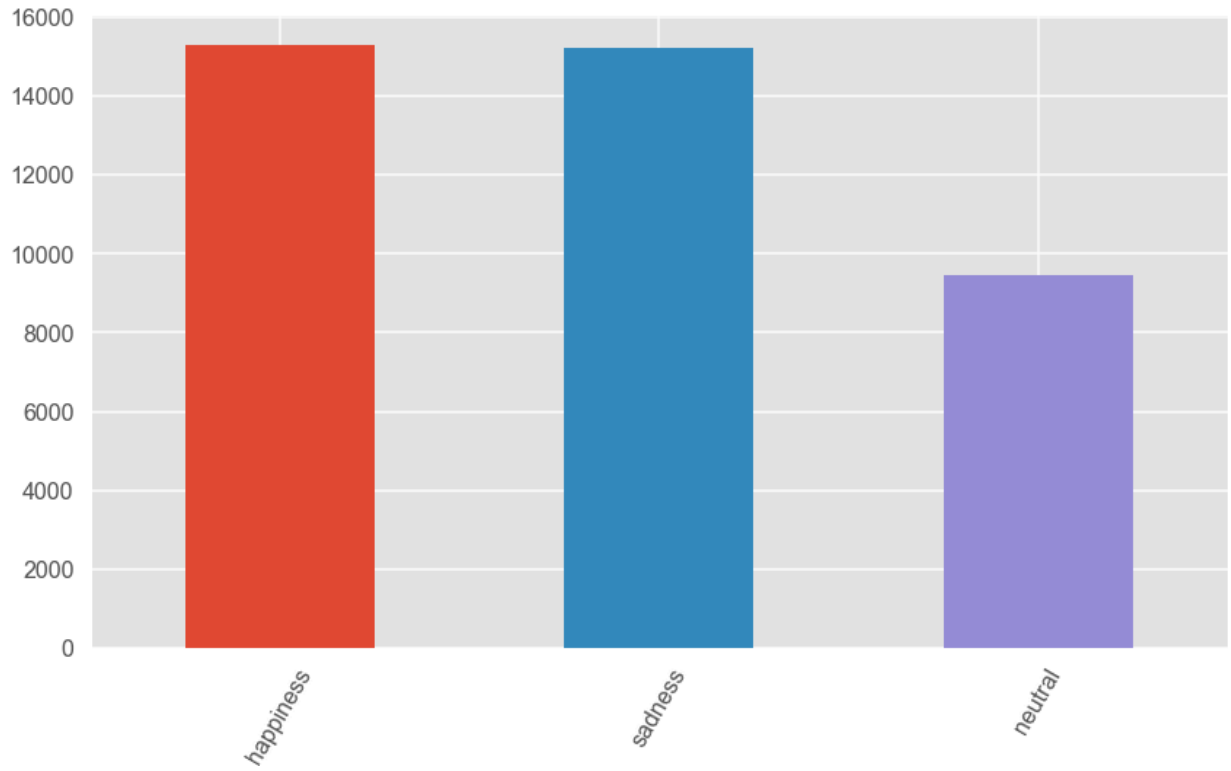
b. Data Visualization and EDA:



Here is the number of tweets who tweeted most in order. The highest number of tweets is 27. This doesn't do serious impact on the data.



This is the number of tweets on specific feelings. There are 13 labels at first.



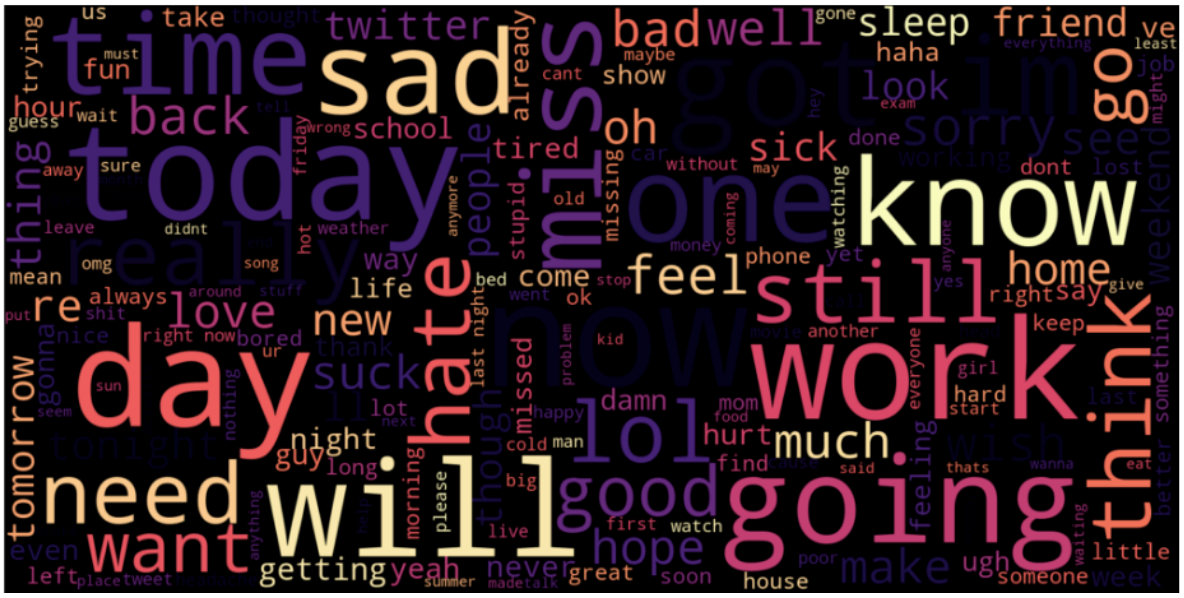
Because it is hard to get a score above 45%, then I shrank the no of labels to 3 as happiness, sadness and neutral.

WordCloud : A **Wordcloud** (or **Tag cloud**) is a visual representation of text data. It displays a list of words, the importance of each beeing shown with font size or color. This format is useful for quickly perceiving the most prominent terms.

- Happy Tweets :

Most prominent words: Happy, thank, day, fun, mother, haha, nice…

- Sad Tweets:

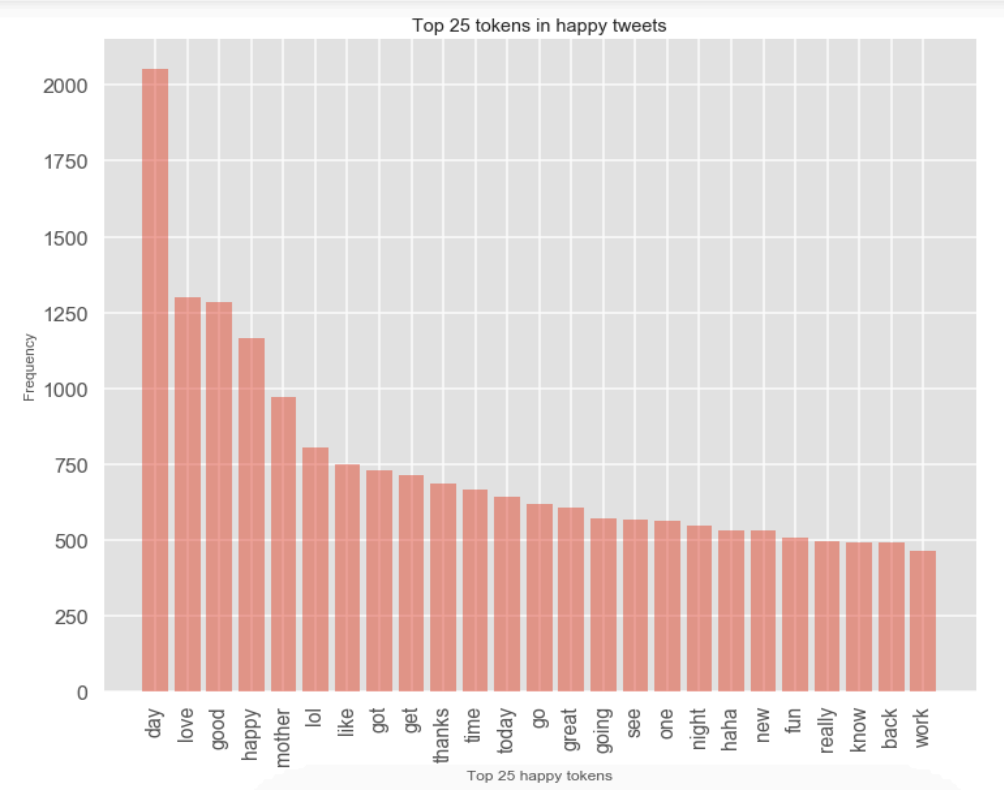


Most prominent words of this kind: sad, today, will, hope, bad, work…

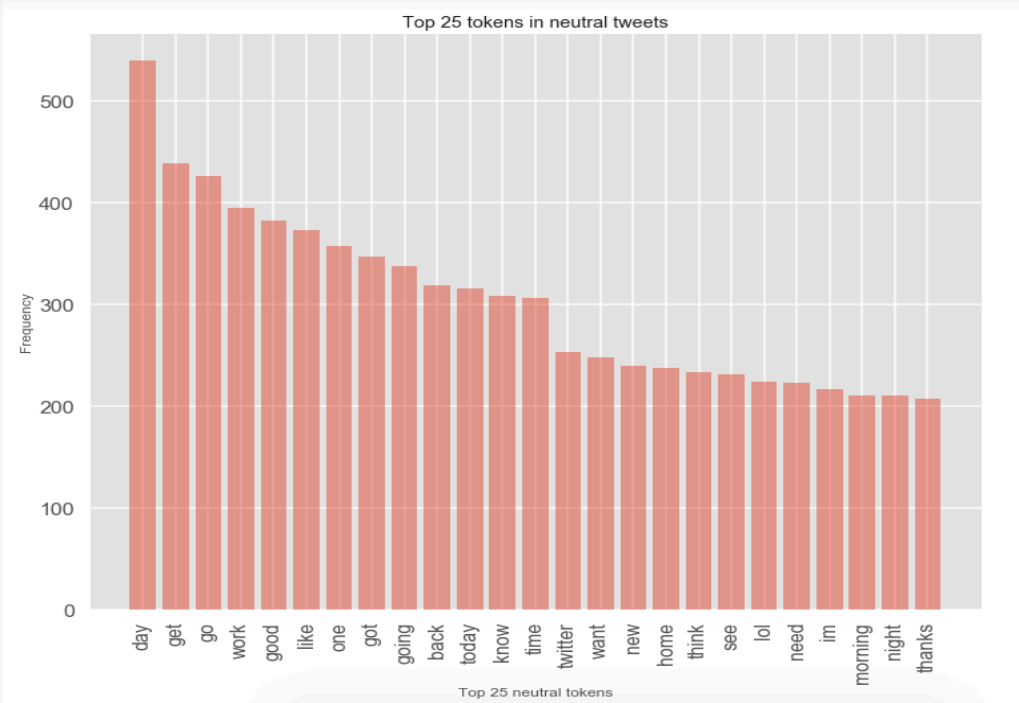
- Neutral Tweets:

Most prominent words: day, know, work, now, twitter, going…..

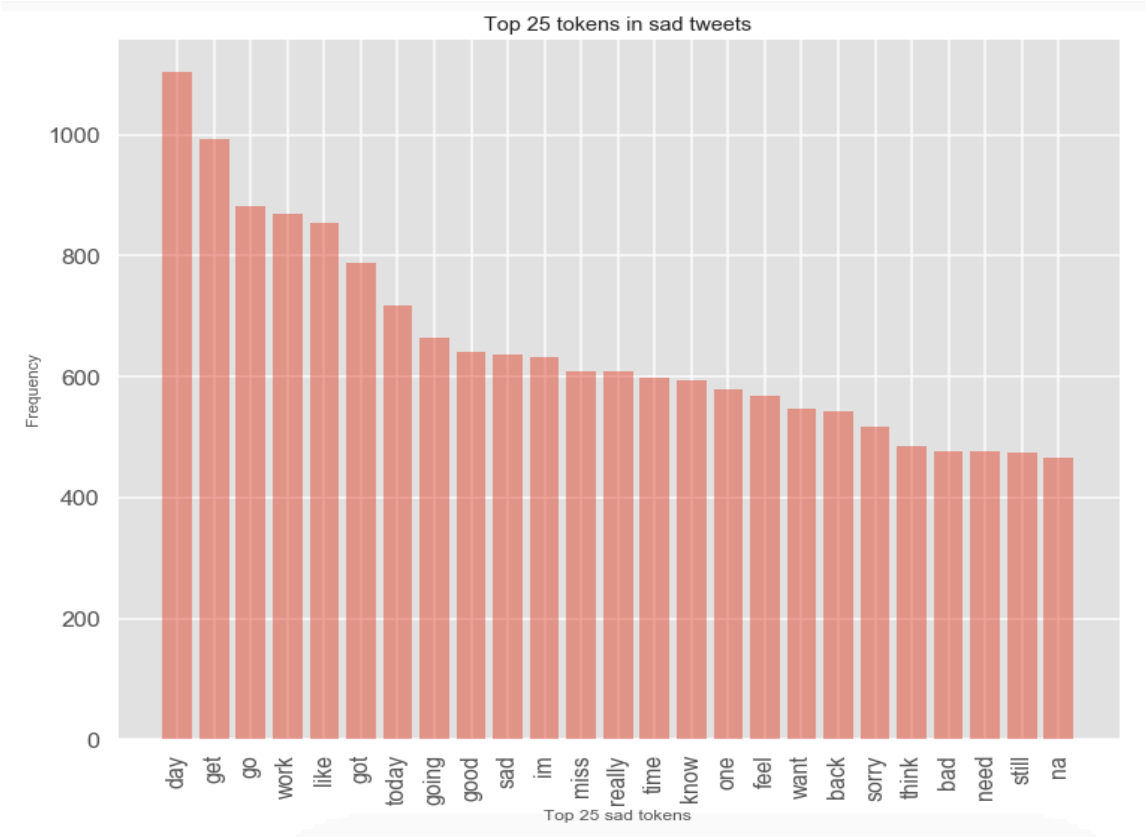
These are the top 25 tokens in happy tweets:



These are the top 25 tokens in sad tweets:



These are the top 25 tokens in neutral tweets:



6. Modelling:

I applied several algorithms such as:

a. Logistic Regression,

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

logreg = LogisticRegression(random_state=0)

logreg.fit(cv_train, y_train)

pred = logreg.predict(cv_test)

metrics.accuracy_score(pred, y_test)
```

```
0.6065333333333334
```

b. Linear SVM,

Linear SVC

```
from sklearn.svm import LinearSVC

Lsvc = LinearSVC()

Lsvc.fit(cv_train, y_train)

pred= Lsvc.predict(cv_test)

metrics.accuracy_score(y_test, pred)
```

```
0.5749333333333333
```

c. Naïve Bayes,

Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB

nb_classifier = MultinomialNB()

nb_classifier.fit(cv_train, y_train)

pred = nb_classifier.predict(cv_test)

metrics.accuracy_score(y_test, pred)
```

```
0.6098
```

d. and others depicted below (TF-IDF):

```
Validation result for Logistic Regression features
accuracy score: 62.22%
Validation result for Linear SVC features
accuracy score: 59.30%
Validation result for LinearSVC with L1-based feature selection features
accuracy score: 59.80%
Validation result for Multinomial NB features
accuracy score: 61.57%
Validation result for Bernoulli NB features
accuracy score: 61.41%
Validation result for Ridge Classifier features
accuracy score: 60.56%
Validation result for AdaBoost features
accuracy score: 55.87%
Validation result for Perceptron features
accuracy score: 54.71%
Validation result for Passive-Aggresive features
accuracy score: 56.85%
Validation result for Nearest Centroid features
accuracy score: 53.91%
```

and

f. Deep Learning Algorithm (Basic) on:

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Embedding, LSTM

model = Sequential()
model.add(Embedding(2500,128,input_length=X.shape[1],dropout=0.2))
model.add(LSTM(300, dropout_U=0.2,dropout_W=0.2))
model.add(Dense(3,activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,optimizer='adam',metrics=['accuracy'])
```

```
model.fit(X_tr ,y_tr, epochs=5,verbose=2,batch_size=32)
```

```
Epoch 1/5
 - 118s - loss: 0.9404 - acc: 0.5573
Epoch 2/5
 - 119s - loss: 0.8653 - acc: 0.6136
Epoch 3/5
 - 118s - loss: 0.8350 - acc: 0.6314
Epoch 4/5
 - 119s - loss: 0.8003 - acc: 0.6504
Epoch 5/5
 - 119s - loss: 0.7635 - acc: 0.6690
```

i. CountVectorizer (bag of words),

ii. Tf-idf Vectorizer,

iii. others such as

1) topic modeling (LDA),

Topic Modeling

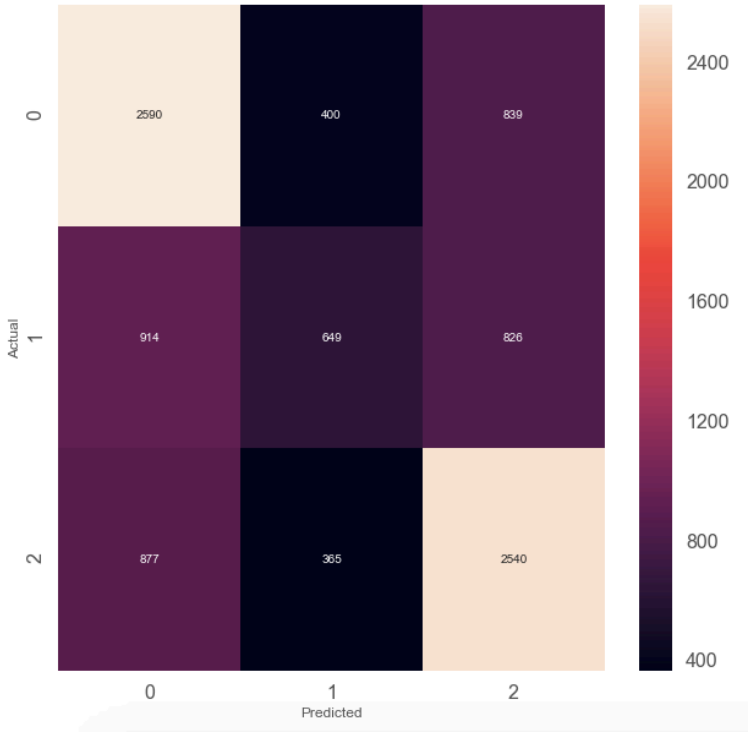
In [85]:
```
from sklearn.decomposition import LatentDirichletAllocation

lda = LatentDirichletAllocation(n_topics=3, max_iter=100, random_state=0)
dt_train = lda.fit_transform(cv_train)
dt_test = lda.transform(cv_test)

features = pd.DataFrame(dt_train, columns=['T1', 'T2', 'T3'])
features
```

Out[85]:

	T1	T2	T3
0	0.041684	0.243554	0.714762
1	0.925560	0.037359	0.037080
2	0.925418	0.037301	0.037282
3	0.026618	0.762926	0.210455
4	0.048124	0.049001	0.902875
5	0.830821	0.085682	0.083498
6	0.028106	0.943740	0.028154
7	0.033676	0.932924	0.033400
8	0.024049	0.101810	0.874142
9	0.066777	0.066699	0.866524
10	0.288042	0.651026	0.060932

Overall my score remained at %63. And this is the confusion matrix below.

Confusion Matrix:



7. Conclusion:

The analysis sought to improve the accuracy of the twitter sentiment analysis. We achieved this by building a statistical model utilizing vectorized features out of twitter text.

I first worked on the preprocessing part of the project by means of several tools such as nltk, beautifulsoup, tokenizers, some python functions.

And then, I considered improving the accuracy score by trying different models. Deep Neural Network gave me the best score from the batch of logistic regression, naïve bayes, decision tree etc.

8. Future Work:

a. Improving the accuracy of this model depends on the number of data/ observations we have. I tried adding more data, however I faced the computational barrier of my personal computer. I will try Cloud Platforms as well as a more powerful computer to avoid this handicap.

b. I knew that NLP would be the most important part of Data Science, and there is much to implement such as Word2Vec, Doc2Vec, Phrase Modelling, POS Tagging etc. to improve modelling.