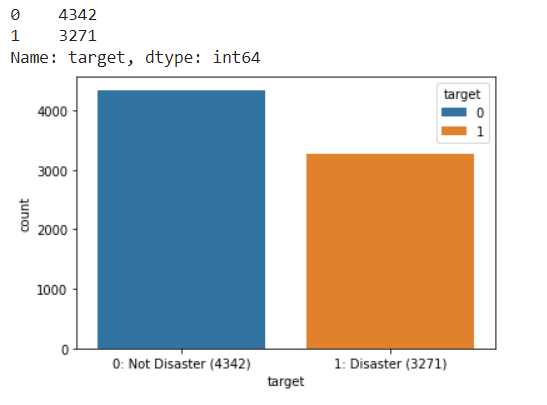
**EDA- Exploratory Data Analysis**

* **NULL values:** 2 dataset Features has null values as Location and Keyword but that are ignored as our main focus is on text and target features in the Data frame.

The first thing we check while working on classification tasks is the target label to know from the beginning whether there is any imbalanced classification task present.

There is an imbalance seen towards the negative class which can be seen from the countplot graph (0 🡪 tweets for not-disaster and 1 🡪 tweets for disaster).

Model prediction will need a balance binary set of target feature, though in our case we will not balance the target feature because it has a small number of changes as (0,1) = (4342, 3271).

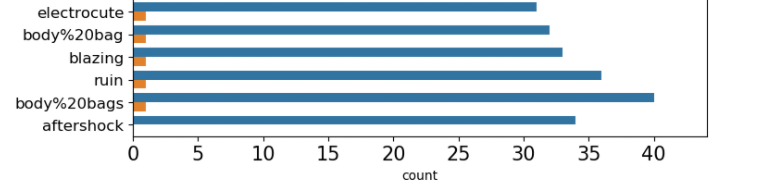
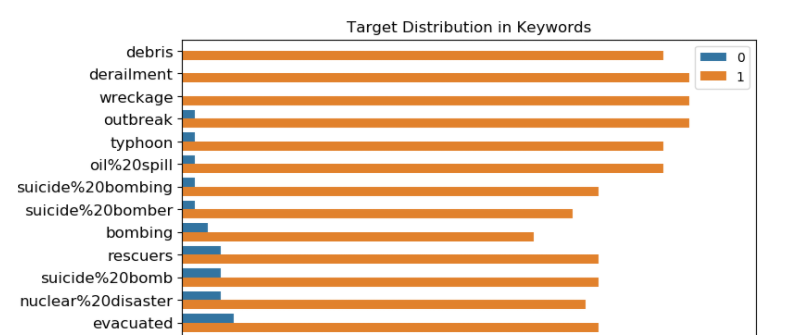
Data Breakdown: Our train set consists of 7613 rows and 5 columns and then dropping duplicate attributes to.

The next thing, in general, we want to know the amount of data that’s missing from each of the features in our dataset.

The number of records with missing location: 2533, 33% missing values

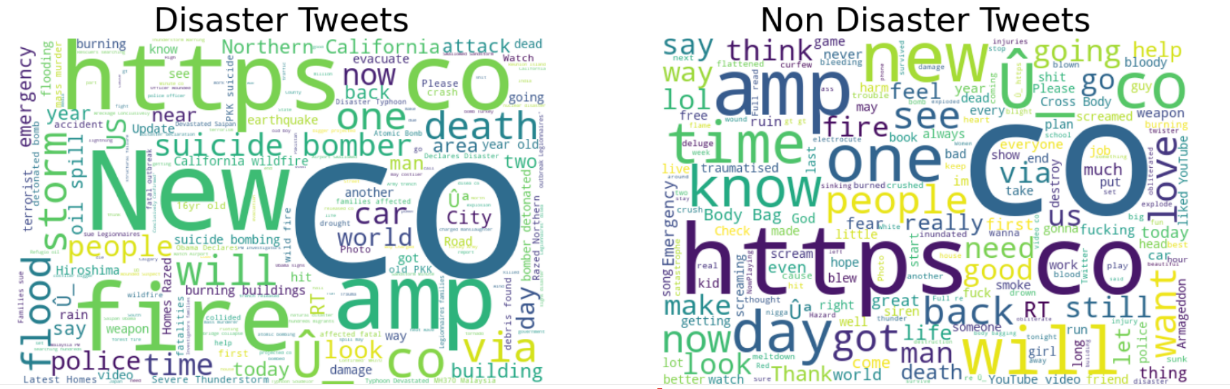
The number of records with missing keywords: 61, 1% missing values

* **Target Distribution in Keywords:**



On the X-axis number of times that keyword is used is represented and on the Y axis values are sorted on 0 of target feature. Values are divided into binary set in 0 and 1, from the image we can see that ‘debris’, ‘derailment’, and ‘wreckage’ keywords are causing several disasters in the tweets which is more than 35 times and on the other hand ‘aftershock’, ‘body%20bags’, and etc. keywords cause lease number of disasters in the tweets feature which is also varying between 30 and 40.

**Word-cloud:** We have used word cloud to distinguish between tweets, as in the tweets containing words related to disaster for example; floods, suicide bombing, police etc.; are separated and contained in ‘word-cloud1’ and the tweets which contains words NOT related example; free, people, love etc.; to disaster are separated and contained in ‘word-cloud2’.



**FEATURE ENGINEERING:**

Tweets require lots of cleaning but it is inefficient to clean every single tweet because that would consume too much time. A general approach must be implemented for cleaning.

* The most common type of words that require cleaning in have punctuations at the start or end. Those words don’t have embeddings because of the trailing punctuations. Punctuations #, @, !, ?, +, &, -, $, =, <, >, |, {, }, ^, ', (, ),[, ], \*, %, ..., ', ., :, ; are separated from words
* Special characters that are attached to words are removed completely
* Contractions are expanded
* Urls are removed
* Character entity references are replaced with their actual symbols
* Typos and slang are corrected, and informal abbreviations are written in their long forms
* Some words are replaced with their acronyms and some words are grouped into one
* Finally, hashtags and usernames contain lots of information about the context but they are written without spaces in between words so they don't have embeddings. Informational usernames and hashtags should be expanded but there are too many of them. I expanded as many as I could, but it takes too much time to run clean function after adding those replace calls.

Distributions of meta features in classes and datasets can be helpful to identify disaster tweets. It looks like disaster tweets are written in a more formal way with longer words compared to non-disaster tweets because most of them are coming from news agencies. Non-disaster tweets have more typos than disaster tweets because they are coming from individual users. The meta features used for the analysis are:

* word\_count number of words in text
* unique\_word\_count number of unique words in text
* stop\_word\_count number of stop words in text
* url\_count number of urls in text
* mean\_word\_length average character count in words
* char\_count number of characters in text
* punctuation\_count number of punctuations in text
* hashtag\_count number of hashtags (**#**) in text
* mention\_count number of mentions (**@**) in text

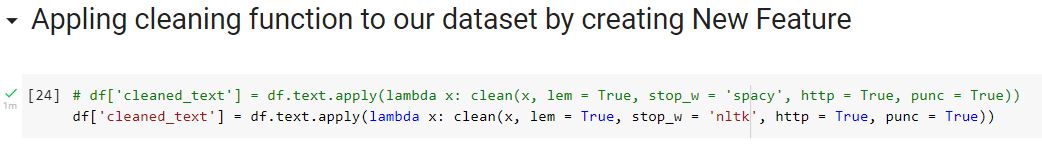
**Removing Stop words:** In our Text-feature we foundsome extremely common words for example; as, is, the, in, of etc.; which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are removed with the help of stop-words.

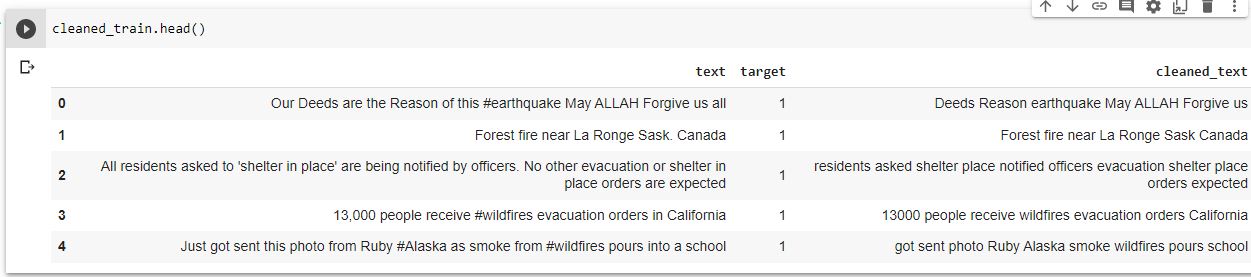
Approx. spacy has 362 english wordlist and nltk has 179 english wordlist

We are using nltk and spacy for removing stop-words, after using both stop-words methods we have came too the conclusion that nltk is more efficient than spacy even though, spacy has more word list in its library.

**Removing Http links:** Deleting patterns of http links from text feature where to remove links we are using regex substitute command.

**Removing Extra-Spaces:** Removing all the Extra spaces that are around words or letters

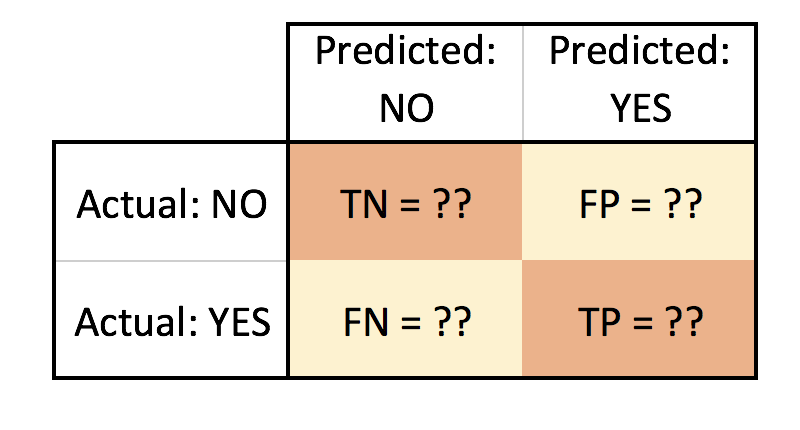




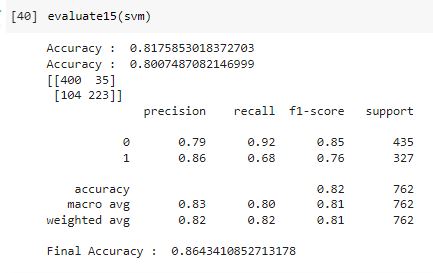
* After Applying Feature Engineering ‘text’ Feature has curtailed to ‘cleaned\_text’ as seen in the image above.
* **Bag of Words:** we are not using bag of words as by using this might decrease our accuracy as it depends on the similar words
* **Test-Train-Split**
* Split arrays or matrices into random train and test subsets

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model. Splitting the Dataset on test size of 0.1 as 10% of the data for testing and 90% dataset for training and keeping shuffle as True and stratify as y for **test set.**

* **TfidfTranformer and TfidfVectorizer**
* then we imported sklearn’s TD-IDF vectorizer, which tokenizes documents, learns the vocabulary and inverse document frequency weightings of the tweets, which allows for the encoding of new documents
* Convert a collection of raw documents to a matrix of TF-IDF features.
* Each sentence is a vector, the sentences you've entered are matrix with 3 vectors. In each vector the numbers (weights) represent features tf-idf score. For example: 'julie': 4 --> Tells you that the in each sentence 'Julie' appears you will have non-zero (tf-idf) weight. As you can see in the 2'nd vector:[ 0. 0.68091856 0. 0. 0.51785612 0.51785612 0. 0. 0. 0. 0. ]
* TF = (Number of times term T appears in the particular row) / (number of terms in that row)
* IDF = log(N/n)
* With the help of TF\*IDF Matrix the text prediction are nearly correct because it analyses the prediction based on number of times the occurrences of words in text feature

**Confusion-Matrix:**

* A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

****

* **Classification Report:** Build a text report showing the main classification metrics.
* Classification report shows various accuracy for both binary classifications.
* Which is very useful to predict the outcome.

**Model Selection**

Not Balancing the data as under sampling or oversampling as target data is almost balances

We have user SVM and naive\_bayes from sklearn to predict the outcome of the classification problem.

SVM:

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples.

The multiclass support is handled according to a one-vs-one scheme.

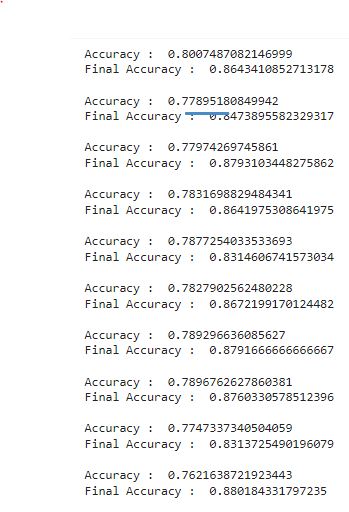
IN svm we are keeping kernel as linear and gamma as auto

Naïve\_bayes:

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem with the “naive” assumption of conditional independence between every pair of features given the value of the class variable. Bayes’ theorem states the following relationship, given class variable and dependent.

This method is very accurate for text classification encoding and predicting the final value

This method is also very connected with Tfidfvectorizer



**Final Average Answer with randomsearch is 86.19%**