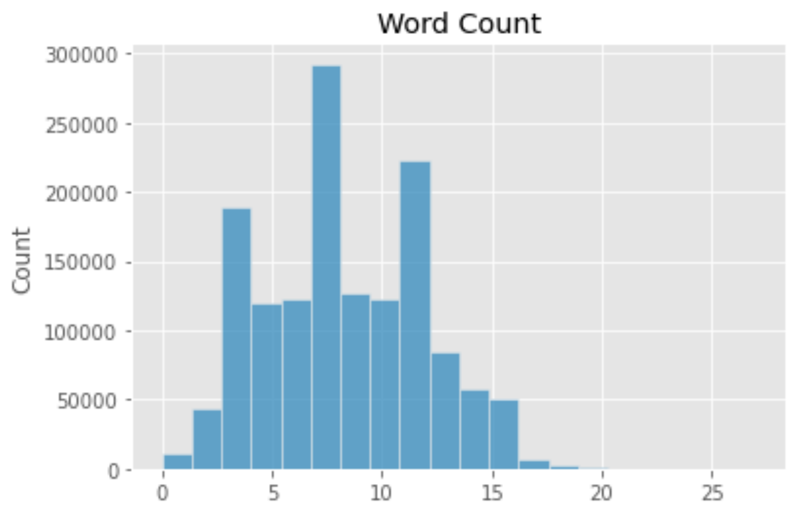
Preprocess :

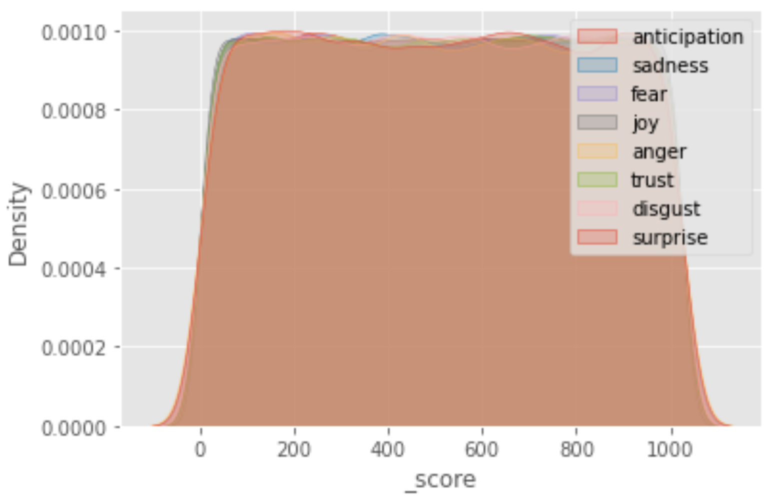
```
def preprocess(message):
    # Lowercase the twit message
    text = message.lower()
    # Remove Punctuation
    text  = "".join([char for char in text if char not in string.punctuation])
    text = re.sub('[0-9]+', '', text)

    # Replace ticker symbols with a space. The ticker symbols are any stock symbol that starts with $.
    text = re.sub('\$[a-zA-Z0-9]*', ' ', text)

    # Replace everything not a letter or apostrophe with a space (will reomve emoji)
    # text = re.sub('[^a-zA-Z\']', ' ', text)
    # Remove single letter words
    text = ' '.join( [w for w in text.split() if len(w)>1] )


    # stem may have problems like 'y' replaced by i: this->thi, pretty->pretti (u can comment out this part)
    porter = PorterStemmer()
    words = text.split()
    text = " ".join([porter.stem(word) for word in words])

    #remove 1h!
    text = text.replace('lh','')

    return text
```

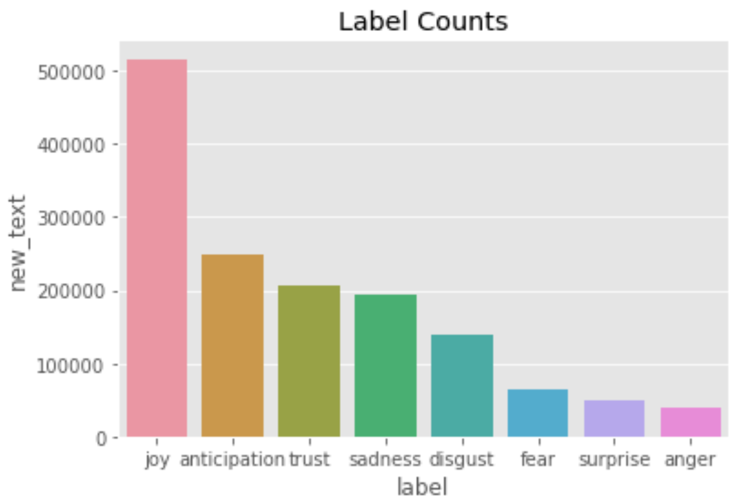
1. I make all word in content lowercase.
2. Remove Punctuation
3. Replace ticker symbols with a space. The ticker symbols are any stock symbol that starts with $.
4. Replace everything not a letter or apostrophe with a space (will also remove emoji)
5. Remove single letter words
6. stem may have problems like 'y' replaced by i: this->thi, pretty->pretti (u can comment out this part)
7. clean the stopwords with NLTK.

```
from nltk.corpus import stopwords

nltk.download('stopwords')
STOPWORDS = set(stopwords.words('english'))
def clean_stopword(article):
    for word in STOPWORDS:
        token = ' ' + word + ' '
        article = article.replace(token, ' ')
        article = article.replace('  ', ' ')
    return article
```

EDA: I am not clear the meaning of score, so I didn't consider it as a feature in the following steps







Naïve Bayes + tf-idf :

I use Naïve bayes classifier and tf-idf vectorizer, my training accuracy is about 0.52 ,and I got score about 0.3xxx close to 0.4.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
mnb_model = MultinomialNB()
mnb_model = mnb_model.fit(X_train, y_train)

y_train_pred_mnb = mnb_model.predict(X_train)
y_test_pred_mnb = mnb_model.predict(X_test)

acc_train_mnb = accuracy_score(y_true=y_train, y_pred=y_train_pred_mnb)


print('training accuracy: {}'.format(round(acc_train_mnb, 2)))
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer()
tfidf_vectorizer.fit(df_data_train['new_text'])
```

LSTM :

Then I tried bidirectional LSTM and the preprocess above, I got score about 0.4xxx, but I didn't try too much on this model because I spent more time on fine tune bert.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 64)          320000

 bidirectional (Bidirectiona  (None, 128)              66048
 l)

 dense (Dense)               (None, 64)                8256

 dense_1 (Dense)             (None, 8)                 520

=================================================================
Total params: 394,824
Trainable params: 394,824
Non-trainable params: 0
_________________________________________________________________
```

Bert :

My last method is to fine tune Bert, and it's a normal fine tune Bert. I used about 200,000 rows of data to train, since if I use the whole data set, the training time will be too long.

Something Interesting I found is that I don't need to do preprocessing before fine tuning Bert. When I did preprocessing before fine tuning Bert, the performance decreased apparently. And I found the reason is that BertTokenizer can process those punctuations and emojis that I get rid of and make them meaningful for training.

My final score is : 0.51371

Tokenizer:

```
# load the tokenizer
tokenizer = BertTokenizerFast.from_pretrained(model_name, do_lower_case=True)
```

My hyperparameters:

```
training_args = TrainingArguments(
    output_dir='./results',          # output directory
    num_train_epochs=4,              # total number of training epochs
    per_device_train_batch_size=8,  # batch size per device during training
    per_device_eval_batch_size=20,   # batch size for evaluation
    warmup_steps=500,                # number of warmup steps for learning rate scheduler
    weight_decay=0.01,               # strength of weight decay
    logging_dir='./logs',            # directory for storing logs
    load_best_model_at_end=True,     # load the best model when finished training (default metric is loss)
    # but you can specify `metric_for_best_model` argument to change to accuracy or other metric
    logging_steps=800,               # log & save weights each logging_steps
    save_steps=800,
    learning_rate = 1e-05,
    evaluation_strategy="steps",     # evaluate each `logging_steps`
)
```