Dataset:

```
https://data.world/crowdflower/sentiment-analysis-in-text
```

In [ ]:

```
import pandas as pd
# Reading the csv dataset into a pandas dataframe
df = pd.read_csv('E:/Internships/TCS-iON/Code/MyCode/Tweets/text_emotion.csv', encoding = '
# Adding a column representing 1 for positive and 0 for negative sentiments
df['senti'] = df.apply(lambda x: 1 if (x['sentiment'] == 'enthusiasm' or x['sentiment'] ==
# Deleting unnecessary columns
df = df.drop(['tweet_id', 'sentiment', 'author'], axis = 1)
# Converting the data type to string
df['content'] = df["content"].astype("str")
# Converting all text to lowercase for use
df['content'] = df['content'].str.lower()
df.head()
```

review	senti
@tiffanylue i know i was listenin to bad habi...	0
layin n bed with a headache ughhhh...waitin o...	0
funeral ceremony...gloomy friday...	0
wants to hang out with friends soon!	1
@dannycastillo we want to trade with someone w...	0

i am going to start reading the harry potter series again because that is one awesome story.

In [ ]:

```
import re
import string
from nltk import WordNetLemmatizer
from nltk.stem.snowball import SnowballStemmer
from nltk.corpus import stopwords
# Initialising the nltk stop_words, stemmer and lemmatizer functions
stop_words = set(stopwords.words("english"))
lemmatizer = WordNetLemmatizer()
stemmer = SnowballStemmer("english")
# Creating a function for text cleaning
def textCleanser(myText):
    # Converting each tweet to string
    myText = str(myText)
    # Removing the name titles and the period symbols after it
    myText = re.sub(r'[mdsr]r(s)?\.', '', myText)
    # Removing the '@username' mentions
    myText = re.sub(r'@\w+\s', '', myText)
    # Removing punctuation
    myPunct = string.punctuation
    punctToSpace = str.maketrans(myPunct, len(myPunct)*' ')
    myText = myText.translate(punctToSpace)
    # Removing urls
    myText = re.sub(r'((http(s?)?)://?)(www\.?).+\.com', '', myText)
    myText = re.sub(r'http(s?)', '', myText)
    # Removing numbers
    myText = re.sub(r'\d+', '', myText)
    # Removing stopwords
    myText = [word for word in myText.split(' ') if not word in stop_words]
    myText = [word for word in myText if word != '']
    # Lemmatizing the text
    myText = [lemmatizer.lemmatize(token) for token in myText]
    # Stemming the text
    # myText = [stemmer.stem(token) for token in myText]
    return myText
for i in range(len(df['content'])):
    df['content'][i] = textCleanser(df['content'][i])
df.head()
```

```
                                          review   senti
 [know, listenin, bad, habit, earlier, started,...      0
   [layin, n, bed, headache, ughhhh, waitin, call]      0
               [funeral, ceremony, gloomy, friday]      0
                         [want, hang, friend, soon]      1
     [want, trade, someone, houston, ticket, one]       0
```

```
['going', 'start', 'reading', 'harry', 'potter', 'series', 'one', 'awesome', 'story']
```

In [ ]:

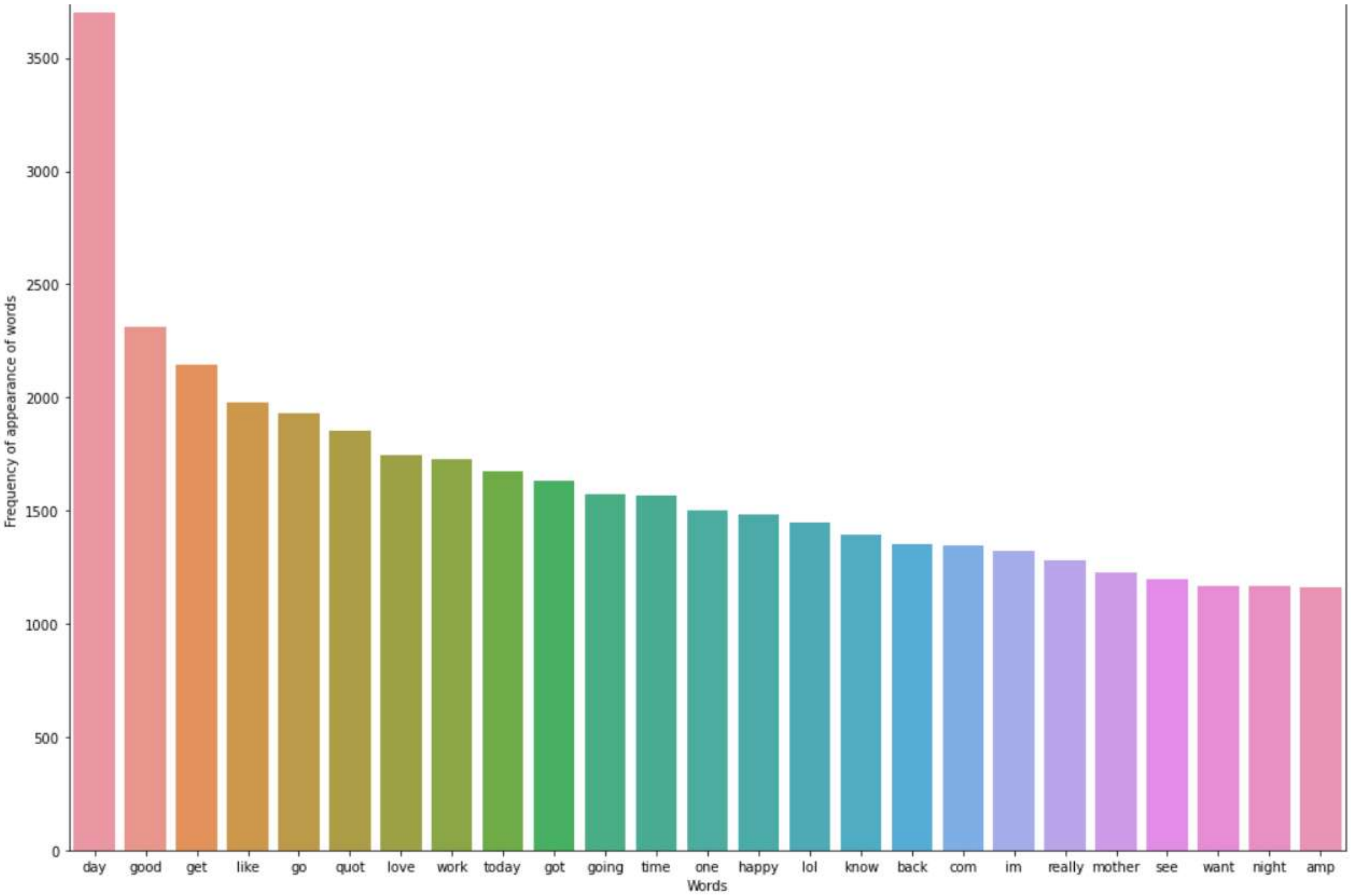
```
myReviews = []
for i in range(len(df['content'])):
    for j in df['content'][i]:
        if j != 'br' and j != 'http':
            myReviews.append(j)
```

In [ ]:

```
from collections import Counter
import collections
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.metrics import classification_report
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
import numpy as np
np.random.seed(1234)
# Initialising the Count Vectorizer
cv = CountVectorizer()
myBow = cv.fit_transform(myReviews)
wordFrequency = dict(zip(cv.get_feature_names(), np.asarray(myBow.sum(axis = 0)).ravel()))
wordCounter = collections.Counter(wordFrequency)
# Storing the frequency of appearance of words
dfWordCounter = pd.DataFrame(wordCounter.most_common(25), columns = ['word', 'frequency'])
```

In [ ]:

```
# Plotting the top 25 most frequently occuring words
plt.close('all')
fig, ax = plt.subplots(figsize = (17, 12))
sns.barplot(x = 'word', y = 'frequency', data = dfWordCounter, ax = ax)
sns.set_palette('pastel')
plt.xlabel('Words')
plt.ylabel('Frequency of appearance of words')
plt.show()
```


Frequency of appearance of words
3500
3000
2500
2000
1500
1000
500
0
day
good
get
like
go
quot
love
work
today
got
going
time
one
happy
lol
know
back
com
im
really
mother
see
want
night
amp
Words

In [ ]:

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense , Input , LSTM , Embedding, Dropout , Activation, GRU, Flatt
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.models import Model, Sequential
from keras.layers import Convolution1D
from keras import initializers, regularizers, constraints, optimizers, layers
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
# Splitting the dataframe to training and testing data
X_train, X_test, y_train, y_test = train_test_split(df["review"], df['senti'],test_size=0.2
# Initialising the tokenizer
max_features = 4000
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(X_train)
list_tokenized_train = tokenizer.texts_to_sequences(X_train)
maxlen = 130
X_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
y = y_train
embed_size = 128
# Initializing a bidirectional sequential LSTM using Adam optimizer, positive activation fu
model = Sequential()
model.add(Embedding(max_features, embed_size))
model.add(Bidirectional(LSTM(32, return_sequences = True)))
model.add(GlobalMaxPool1D())
model.add(Dense(20, activation="relu"))
model.add(Dropout(0.05))
model.add(Dense(1, activation="sigmoid"))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
batch_size = 100
epochs = 5
model.fit(X_t,y, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

Epoch 1:

```
time: 75s
speed: 292ms/step
loss: 0.5886
accuracy: 0.6948
val_loss: 0.5567
val_accuracy: 0.7227
```

Epoch 2:

```
time: 59s
speed: 229ms/step
loss: 0.5116
accuracy: 0.7575
val_loss: 0.5612
val_accuracy: 0.7181
```

Epoch 3:

```
time: 60s
speed: 234ms/step
loss: 0.4794
accuracy: 0.7789
val_loss: 0.5857
val_accuracy: 0.7094
```

Epoch 4:

```
time: 69s
speed: 269ms/step
loss: 0.4484
accuracy: 0.7964
val_loss: 0.5969
val_accuracy: 0.7023
```

Epoch 5:

```
time: 69s
speed: 271ms/step
loss: 0.4111
accuracy: 0.8214
val_loss: 0.6331
val_accuracy: 0.7073
```

In [ ]:

```
# Testing the model
list_sentences_test = X_test
list_tokenized_test = tokenizer.texts_to_sequences(list_sentences_test)
X_te = pad_sequences(list_tokenized_test, maxlen=maxlen)
prediction = model.predict(X_te)
y_pred = (prediction > 0.5)
from sklearn.metrics import f1_score, confusion_matrix
print('F1-score: {0}'.format(f1_score(y_pred, y_test)))
print('Confusion matrix:')
confusion_matrix(y_pred, y_test)
```

F1-score:

```
0.5906515580736543
```

Confusion matrix:

```
[[4020, 1400],
 [ 912, 1668]]
```

Accuracy:

```
71.10%
```