```
## General Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
from wordcloud import WordCloud ### For visualising the frequent words
import re
import random
import string
import math
import itertools

# To mute general warnings
import warnings
warnings.filterwarnings('ignore')

## For data preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer

## For building our Model
import tensorflow as tf
from tensorflow.keras.layers import Input,Dense,LSTM,Embedding,Conv1D,Bidirectional,SpatialDropout1D,Dropout
from tensorflow.keras import Sequential
from tensorflow.keras.callbacks import Callback,ModelCheckpoint,CSVLogger,ReduceLROnPlateau,LearningRateScheduler,EarlyStopping
from tensorflow.keras.optimizers import Adam

## For creating vocabulary dictionary:
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

## For model evaluation:
from sklearn.metrics import confusion_matrix,classification_report

## For Text Processing:
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer,WordNetLemmatizer
from nltk.tokenize import word_tokenize
!pip install pyspellchecker
from spellchecker import SpellChecker
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyspellchecker
  Downloading pyspellchecker-0.7.1-py3-none-any.whl (2.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 6.4 MB/s eta 0:00:00
Installing collected packages: pyspellchecker
Successfully installed pyspellchecker-0.7.1
```

```
np.random.seed(31415)
tf.random.set_seed(2)
```

```
plt.style.use('fivethirtyeight')
```

```
# Import the google drive folders that contain the data
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
%cd /content/drive/MyDrive/DSC680/Weeks1-4/Week2/data/
```

```
/content/drive/MyDrive/DSC680/Weeks1-4/Week2/data
```

```
%ls
```

```
antidepressant/   depressed/                            hopeless/
antidepressants/  depression/                           lonely/
archive/          depression_dataset_reddit_cleaned.csv/ suicide/
```

```
df = pd.read_csv('archive/training.1600000.processed.noemoticon.csv', encoding='latin',header=None)
df.head()
```

	0	1	2	3	4	5
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53	NO_QUERY	mattycus	@Kenichan I dived many

```
#Get shape of dataframe
print(f"ROWS: {df.shape[0]} \nColumns:{df.shape[1]}")
```

```
ROWS: 1600000
Columns:6
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600000 entries, 0 to 1599999
Data columns (total 6 columns):
 #   Column  Non-Null Count    Dtype
---  ------  --------------    -----
 0   0       1600000 non-null  int64
 1   1       1600000 non-null  int64
 2   2       1600000 non-null  object
 3   3       1600000 non-null  object
 4   4       1600000 non-null  object
 5   5       1600000 non-null  object
dtypes: int64(2), object(4)
memory usage: 73.2+ MB
```

```
#Get information on the columns
df.columns
```

```
Int64Index([0, 1, 2, 3, 4, 5], dtype='int64')
```

```
#Rename columns
df.columns = ['sentiment', 'id', 'date', 'query', 'user_id', 'text']
```

```
#value counts of sentiment
df["sentiment"].value_counts()
```

```
0    800000
4    800000
Name: sentiment, dtype: int64
```

Sentiment Distrobution

```
df = df.groupby('sentiment').apply(lambda x: x.sample(n=50000)).reset_index(drop = True)
df.sentiment.value_counts()
```

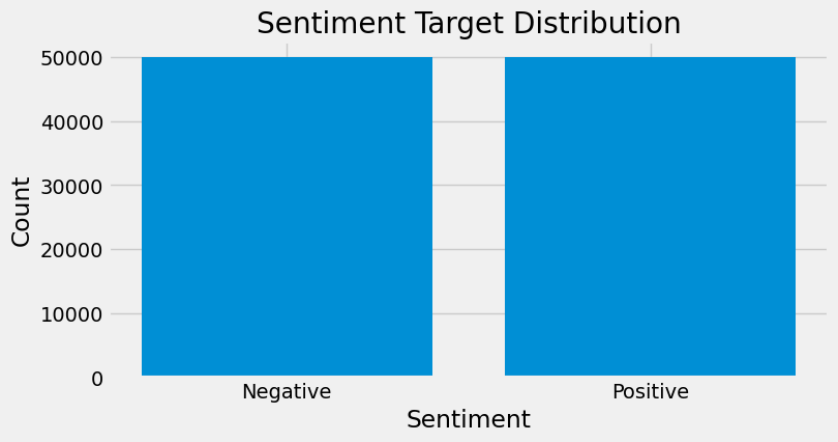
```
0    50000
4    50000
Name: sentiment, dtype: int64
```

```
lab_to_sentiment = {0:"Negative", 4:"Positive"}
df.sentiment = df.sentiment.apply(lambda x: lab_to_sentiment[x])
df.head()
```

sentiment	id	date	query	user_id	text
		Sat Jun 20			@carol_carter1 I

```
val_count = df.sentiment.value_counts()

plt.figure(figsize=(8,4))
plt.bar(val_count.index, val_count.values)
plt.xlabel("Sentiment")
plt.ylabel("Count")
plt.title("Sentiment Target Distribution")
```

```
Text(0.5, 1.0, 'Sentiment Target Distribution')
```




```
### randomly selecting rows and printing them
ind=np.random.randint(0,1000,(10))
df.iloc[ind].text
```

```
842    The Veronicas were great! But its raining now ...
717    just got home after 2 classes. had a great tim...
996         @Anna_Montanna Yeah but it isn't activated.
690    Cub run with sammers before she leaves for ric...
565      @Dunncity  It wasn't suppose to be that easy!!!
515    I better have something in the post....been 6 ...
687    finally get to use internet after 5 days. inte...
810      Blah moving sucks. Why do I have so much stuff
47                            Ugh I'm not tired at all
506    @Clumsyflic the kid that got a second chance s...
Name: text, dtype: object
```

```
## Getting a list of stop words in English:
l_remove="don, don't, ain, aren, aren't, couldn, couldn't, didn, didn't, doesn, doesn't, hadn, hadn't, hasn, hasn't, haven, haven't, isn, isr
stop_words=[i for i in stopwords.words('english') if i not in l_remove.split(", ")]

#### Pattern removing Hyperlinks and Mentions:
pattern = "(@\S+)|(http|ftp|https):\/\/([\w_-]+(?:(?:\.[\w_-]+)+))([\w.,@?^=%&:\/~+#-]*[\w@?^=%&\/~+#-])"

## List of Punctuation marks:
punct=string.punctuation

## Setting a Stemmer Object:
stemmer = SnowballStemmer('english')

## Function to remove numbers:
def remove_numb(text):
    return re.sub("[0-9]+","",text)


### Function for spelling-checker:
spell = SpellChecker()
def correct_spellings(text):
    corrected_text = []
    misspelled_words = spell.unknown(text.split())
    for word in text.split():
        if word in misspelled_words:
```

```
            corrected_text.append(spell.correction(word))
        else:
            corrected_text.append(word)
    ## Checks whether the corrected string is None, meaning there is no proper
    ## spelling available.
    ct=[i for i in corrected_text if i is not None]
    if " ".join(corrected_text):
        return text
    else:
        return " ".join(corrected_text)


### Stop words offered by NLTK:
", ".join(stopwords.words('english'))
```

```
'i, me, my, myself, we, our, ours, ourselves, you, you're, you've, you'll, you'd,
your, yours, yourself, yourselves, he, him, his, himself, she, she's, her, hers,
herself, it, it's, its, itself, they, them, their, theirs, themselves, what, whic
h, who, whom, this, that, that'll, these, those, am, is, are, was, were, be, bee
n, being, have, has, had, having, do, does, did, doing, a, an, the, and, but, if,
```

```
### Viewing the number of rows having MENTIONS:
v=[]
for i in df.text:
    v.append(re.findall(r'(@\S+)',i))

v1=[i for i in v if len(i)!=0]
print(f"There are {len(v1)} rows with Mentions(@yyy) in their text")
```

```
There are 46180 rows with Mentions(@yyy) in their text
```

```
v1[:10]
```

```
[['@carol_carter1'],
 ['@nursebc1974'],
 ['@Maddieeann'],
 ['@MSWindows'],
 ['@tommcfly'],
 ['@KristinaDeFonte'],
 ['@afylayouts'],
 ['@Leanne0710'],
 ['@Hetty4Christ'],
 ['@MARCUS_KENNY']]
```

```
### Viewing the number of rows having HYPERLINKS:
v=[]
for i in df.text:
    v.append(re.findall(r'(http|ftp|https):\/\/([\w_-]+(?:(?:\.[\w_-]+)+))([\w.,@?^=%&:\/~+#-]*[\w@?^=%&\/~+#-])',i))

v2=[i for i in v if len(i)!=0]
print(f"There are {len(v2)} rows with Hyperlinks(http://yyy) in their text")
```

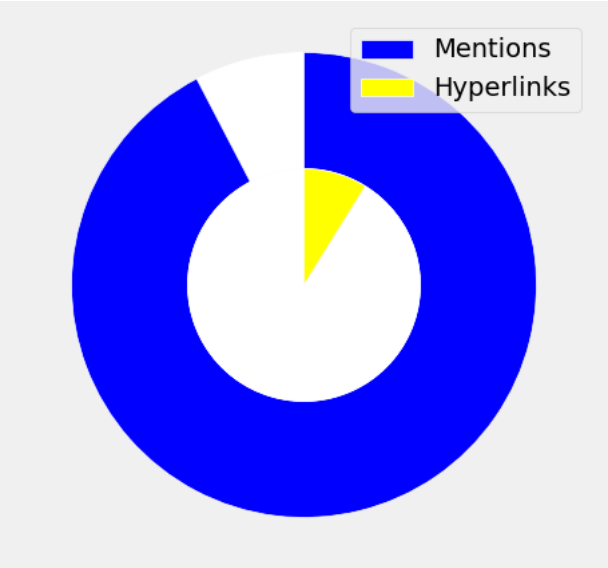
```
There are 4424 rows with Hyperlinks(http://yyy) in their text
```

```
v2[:10]
```

```
[[('http', 'plurk.com', '/p/xhfhf')],
 [('http', 'myloc.me', '/5ABP')],
 [('http', 'twitpic.com', '/7tsbt')],
 [('http', 'plurk.com', '/p/x88wx')],
 [('http', 'twitpic.com', '/6ou3m')],
 [('http', 'myloc.me', '/4NiF')],
 [('http', 'tinyurl.com', '/dy7lcy')],
 [('http', 'myloc.me', '/2IEY')],
 [('http', 'myloc.me', '/2QXk')],
 [('http', 'tr.im', '/oMSv')]]
```

```
labels = ["Mentions","Hyperlinks"]
data = [len(v1),len(v2)]
#number of data points
n = len(data)
#find max value for full ring
k = 10 ** int(math.log10(max(data)))
m = k * (1 + max(data) // k)
```

```
#radius of donut chart
r = 1
#calculate width of each ring
w = r / n

#create colors along a chosen colormap
colors = ["blue","yellow"]

#create figure, axis
fig, ax = plt.subplots(figsize=(5,5))
ax.axis("equal")

#create rings of donut chart
for i in range(n):
    #hide labels in segments with textprops: alpha = 0 - transparent, alpha = 1 - visible
    innerring, _ = ax.pie([m - data[i], data[i]], radius = r - i * w, startangle = 90, labels = ["", labels[i]], labeldistance = 1 - 1 / (1.5
    plt.setp(innerring, width = w, edgecolor = "white")

plt.legend()
plt.show()
```




```
def funct_clean(x,stem=False):
    ### Removing Mentions and Hyperlinks:
    sentence = re.sub(pattern, ' ', str(x).lower()).strip()
    ### Converting to lower case:
    sentence = sentence.lower()
    ### Removing punctuation marks:
    sentence=sentence.translate(str.maketrans('', '', punct))
    ### Tokenising:
    words = word_tokenize(sentence)
    tokens = []
    for token in words:
        if token not in stop_words:   ### Checking whether the token is a stop word
            if stem:
                tokens.append(stemmer.stem(token))  ### Stemming the token(Less time taken)
            else:
                tokens.append(token)

    ## Removing numbers
    t1=remove_numb(" ".join(tokens)).split()
    ## can include the speller function while returning
    return t1


import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
%%time
```

```
df["text_pre"] = df.text.apply(lambda x: funct_clean(x,1))
```

```
CPU times: user 30.9 s, sys: 147 ms, total: 31 s
Wall time: 34.3 s
```

```
data=df.iloc[:1000].copy()
data[["text","text_pre"]]
```

	text	text_pre
0	@carol_carter1 I don't think so I believe it ...	[dont, think, believ, one, time]
1	@nursebc1974 I know!!! My boys r wide awake, ...	[know, boy, r, wide, awak, n, im, fallin, asleep]
2	@Maddieeann either he didn't mean to send it t...	[either, didnt, mean, send, hes, play, hard, get]
3	@MSWindows I really wish microsoft would stop ...	[realli, wish, microsoft, would, stop, push, s...
4	@tommcfly I still think that brazilian's fans ...	[still, think, brazilian, fan, better, alredi,...
...	...	...
995	and when I got out my phone was dead :[[it li...	[got, phone, dead, live, week, ughh]
996	@Anna_Montanna Yeah but it isn't activated.	[yeah, isnt, activ]
997	Aww i'm at retarded leadership shit	[aww, im, retard, leadership, shit]
998	@cehouck that's because we're like the same pe...	[that, like, person, ahaha, ps, messag, wed, p...

```
df.columns
```

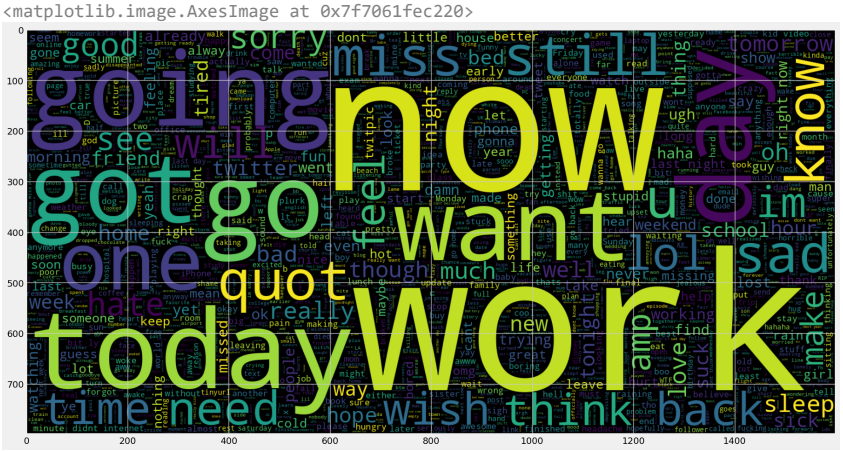
```
Index(['sentiment', 'id', 'date', 'query', 'user_id', 'text', 'text_pre'], dtype='object')
```

```
## Top words from Positive Sentiment Tweets
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(df[df.sentiment == 'Positive'].text))
plt.imshow(wc , interpolation = 'bilinear')
```

```
<matplotlib.image.AxesImage at 0x7f706003edc0>
```

```
## Top words from Negative Sentiment Tweets
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(df[df.sentiment == 'Negative'].text))
plt.imshow(wc , interpolation = 'bilinear')
```

```
<matplotlib.image.AxesImage at 0x7f7061fec220>
```



```
MAX_NB_WORDS = 10000
MAX_SEQUENCE_LENGTH = 30
train_data, test_data = train_test_split(df[["sentiment","text_pre"]], test_size=0.2,
                                         random_state=7) # Splits Dataset into Training and Testing set
print("Train Data size:", len(train_data))
print("Test Data size", len(test_data))
```

```
Train Data size: 80000
Test Data size 20000
```

```
train_data.head(20)
```

	sentiment	text_pre
91526	Positive	[need, work, word, essay, chapter]
19416	Negative	[etsywiki, come, get, error, page]
99079	Positive	[absolut, blast, dont, rememb, harem, rememb, ...
45086	Negative	[final, exam]
54659	Positive	[lol, queen, victoria, precis]
50266	Positive	[still, unhappi, lack, guitar, string, realli,...
62172	Positive	[word, hannah, montana, nobodi, perfect]
66637	Positive	[like, cover, environment, scienc, book, haha]
95845	Positive	[ahhh, kim, kardashian, absolutelyy, love, any...
98862	Positive	[dont, feel, good, eat, dri, cheerio, like, li...
60792	Positive	[dank, vella, kaaka, bubbl, enjoy, page]
54668	Positive	[yay, work]
14560	Negative	[id, love, bake, cooki, current, id, much, rat...
13759	Negative	[that, true, us, men, cant, lol, dont, know, l...
21235	Negative	[hannah, go, well, know, dom, met, katherin, g...
57081	Positive	[ultraviolet, stiff, dylan]
72871	Positive	[cant, wait, till, new, season, ampamp, love, ...
12941	Negative	[twitter, virgin, berri, broke]
45904	Negative	[got, spend, morn, er, cat, cut, eyelid, back,...
62068	Positive	[havin, round, golf, glorious, day]

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_data.text_pre)
word_index = tokenizer.word_index
vocab_size = len(tokenizer.word_index) + 1
print("\nVocabulary Size :", vocab_size)
```

```
Vocabulary Size : 46834
```

```
for i,(j,k) in enumerate(word_index.items()):
    print(j,"-------->",k)
    if i==10:
        break
```

```
im --------> 1
go --------> 2
get --------> 3
day --------> 4
good --------> 5
work --------> 6
like --------> 7
love --------> 8
got --------> 9
dont --------> 10
today --------> 11
```

```
def check_(to_check):
    print(f"-------------ROW {to_check}:---------------")
    print(f"The Train data is: {train_data.text_pre.iloc[to_check]}")
    print(f"The Vectorised Equivalent is: {tokenizer.texts_to_sequences(train_data.text_pre)[to_check]}\n")

check_(0)
check_(5)
```

```
-------------ROW 0:---------------
The Train data is: ['need', 'work', 'word', 'essay', 'chapter']
The Vectorised Equivalent is: [32, 6, 360, 1212, 2052]

-------------ROW 5:---------------
The Train data is: ['still', 'unhappi', 'lack', 'guitar', 'string', 'realli', 'wan', 'na', 'play', 'oh', 'well', 'hope', 'someon', 'exc
The Vectorised Equivalent is: [28, 3270, 1248, 802, 2892, 23, 102, 35, 86, 38, 27, 30, 171, 154, 37, 1004]
```

```
x_train = pad_sequences(tokenizer.texts_to_sequences(train_data.text_pre),
                        maxlen = MAX_SEQUENCE_LENGTH)
x_test = pad_sequences(tokenizer.texts_to_sequences(test_data.text_pre),
                       maxlen = MAX_SEQUENCE_LENGTH)

print("Training X Shape:",x_train.shape)
print("Testing X Shape:",x_test.shape)
```

```
Training X Shape: (80000, 30)
Testing X Shape: (20000, 30)
```

```
encoder = LabelEncoder()
encoder.fit(train_data.sentiment.to_list())

y_train = encoder.transform(train_data.sentiment.to_list())
y_test = encoder.transform(test_data.sentiment.to_list())

y_train = y_train.reshape(-1,1)
y_test = y_test.reshape(-1,1)

print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
y_train shape: (80000, 1)
y_test shape: (20000, 1)
```

```
%%time

!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
```

```
--2023-04-02 18:52:52--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2023-04-02 18:52:52--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2023-04-02 18:52:52--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.02MB/s    in 2m 39s

2023-04-02 18:55:32 (5.17 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]

Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
CPU times: user 2.66 s, sys: 386 ms, total: 3.05 s
Wall time: 3min 25s
```

```
glove_path = 'glove.6B.300d.txt'
embedding_dim = 300
model_path = 'working/best_model.hdf5'
```

```
embeddings_index = {}

### Processing the file
for line in open(glove_path):
```

```
        values = line.split()
        word = value = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

    print('Found %s word vectors.' %len(embeddings_index))
```

```
     Found 400000 word vectors.
```

```
    embedding_matrix = np.zeros((vocab_size, embedding_dim))
    for word, i in word_index.items():  ### unpacking the dictionary
        ### Checking whether the word exists in the pretrained embeddings_index
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector


    def funct_view_embedding(word):
        print(f"For the word \"{word}\", its respective vector representation in a 300 Dimensional Space is:")
        print("\n",embedding_matrix[word_index[f"{word}"]])

    funct_view_embedding("love")  ## since "love" is present in the vocabulary
```

```
       2.14210004e-01  3.31970006e-01 -3.42990011e-01 -4.87349987e-01
       2.22640000e-02  2.78620005e-01  2.38810003e-01  9.77939963e-02
       3.80230010e-01 -3.77440006e-02 -4.19660002e-01 -1.91450000e-01
      -9.58300009e-02  2.68709987e-01  5.28760016e-01 -2.68700004e-01
      -3.44500005e-01  2.54130006e-01  1.36059999e-01  3.65280002e-01
       8.19600001e-02 -5.22239983e-01  3.41590010e-02  1.70190006e-01
      -1.05200000e-01 -8.08730006e-01 -1.80950001e-01 -6.08230010e-02
       2.60140002e-01 -7.87170008e-02 -1.51610002e-01 -6.84379995e-01
       2.18089998e-01  4.73299995e-02 -1.21469997e-01 -3.38919997e-01
      -1.86289996e-02  2.45460004e-01  2.73739994e-01 -2.70449996e-01
      -8.72329995e-02 -4.78709996e-01  1.96300000e-01  2.31240001e-02
       1.94529995e-01 -4.68600005e-01  4.44990009e-01  1.83599994e-01
      -1.64800003e-01  3.85980010e-01  3.60700011e-01 -3.11080009e-01
      -3.46269995e-01  1.90040007e-01 -9.86829996e-02  3.38209987e-01
      -1.39780000e-01 -7.27230012e-01 -1.06600001e-01  1.92079996e-03
      -3.30929995e-01  3.51170003e-01  1.54909998e-01  1.71499997e-01
       2.89330006e-01 -4.92129996e-02 -5.05670011e-01 -2.35129997e-01
      -2.80050009e-01 -2.84869999e-01 -2.43929997e-01 -4.08379994e-02
      -2.72229999e-01  9.45639983e-02  1.73319995e-01  1.86910003e-01
      -1.35849997e-01 -4.64390010e-01 -4.56770003e-01  9.78899971e-02
       3.91130000e-02  1.79010004e-01 -4.56290007e-01  4.75199997e-01
      -2.03879997e-01  2.41229996e-01  6.75509989e-01  6.21050000e-02
      -1.89400002e-01  1.50539994e-01  1.58749998e-01 -2.33649999e-01
       3.74289989e-01 -2.34630004e-01  4.06919986e-01  1.40349999e-01
      -4.29710001e-01  5.67130029e-01 -2.67060012e-01 -6.80280030e-02
      -5.12639999e-01 -3.09450001e-01 -3.90650004e-01 -2.70040005e-01
      -1.18019998e+00  6.64799988e-01 -2.43660003e-01  3.91829997e-01
      -2.89700001e-01 -1.88390002e-01 -4.92819995e-01  1.45380005e-01
       2.44670004e-01 -4.43399977e-03  2.32649997e-01  7.45439976e-02
      -3.00060004e-01 -3.02720010e-01 -1.23939998e-01  3.34729999e-01
       3.38800013e-01  9.74450037e-02 -3.37729990e-01 -5.43160021e-01
      -4.75140005e-01 -1.56959996e-01 -9.35159981e-01 -8.70340019e-02
      -2.67430007e-01  6.46409986e-04  3.19400012e-01 -6.25619991e-03
       1.58539999e+00  1.24839999e-01  4.84809995e-01  7.53939971e-02
       1.89630002e-01 -1.02260001e-01  4.74130005e-01  7.04020023e-01
      -6.44180030e-02 -1.01810005e-02 -6.86190009e-01  1.53350001e-02
       4.78510000e-02  3.76500010e-01  1.04869999e-01  2.29739994e-01
       4.53520000e-01  3.14819992e-01  5.88800013e-02  6.70960024e-02
       1.56790003e-01  1.30989999e-01  3.45810018e-02 -7.38959983e-02
      -3.64329994e-01 -1.88470006e-01  4.15560007e-02 -2.01240003e-01
      -7.39179999e-02  3.87519985e-01  1.67590007e-01 -4.06269997e-01
      -1.32200001e-02 -7.17930019e-01 -2.29039997e-01  2.04740003e-01
      -1.36480004e-01  3.77790004e-01 -4.00029987e-01 -5.01060002e-02
      -3.77990007e-01  7.21089989e-02  2.30570007e-02  2.38790005e-01
      -2.24230006e-01 -8.48340020e-02 -7.07260013e-01 -1.61819994e-01
       2.63729990e-01  1.22259997e-01  8.02820027e-02  7.59629980e-02
      -3.46949995e-01  3.77029985e-01  3.66120011e-01 -9.84160006e-02
       4.74000007e-01 -2.50849992e-01  1.86409995e-01 -2.59930015e-01
       3.90349999e-02 -6.13940001e-01  1.47210002e-01 -6.76190019e-01
      -1.24890000e-01  3.80309999e-01 -2.80609995e-01 -3.96800011e-01
       2.41650008e-02  9.87059996e-02  2.48380005e-01 -4.58810002e-01
       2.03099996e-01 -4.35149997e-01 -7.78040010e-03 -2.46420000e-02
      -1.32040000e+00 -4.10780013e-01  2.81450003e-01  2.81879995e-02
      -2.04699993e-01  1.60370007e-01 -1.66960001e-01  1.37610003e-01
      -3.67139995e-01  1.38009995e-01 -2.06880003e-01  3.82739991e-01
       2.12889999e-01 -7.50010014e-02 -5.03669977e-01 -2.92949993e-02
      -2.12539993e-02 -2.42530003e-01  3.35299999e-01 -3.55340004e-01
       2.53580004e-01  3.89060006e-02  2.43139997e-01 -2.86960006e-02]
```

```
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in word_index.items():  ### unpacking the dictionary
    ### Checking whether the word exists in the pretrained embeddings_index
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector


embedding_matrix.shape
```

```
(46834, 300)
```

```
embedding_layer = Embedding(vocab_size,embedding_dim,
                                      weights=[embedding_matrix],
                                        input_length=MAX_SEQUENCE_LENGTH,
                                        trainable=False)


## Learning Rate:
lr = 1e-2

## Creating the Layers
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedding_sequences = embedding_layer(sequence_input)
x = SpatialDropout1D(0.2)(embedding_sequences)
x = Conv1D(64, 5, activation='relu')(x)
x = Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0.2))(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
outputs = Dense(1, activation='sigmoid')(x)
model = tf.keras.Model(sequence_input, outputs) ## Model Object with input and output layers

## using the standard Adam as the optimiser(can be tweaked according to the user)
model.compile(optimizer=Adam(learning_rate=lr), loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 30)]              0

 embedding (Embedding)       (None, 30, 300)           14050200

 spatial_dropout1d (SpatialD  (None, 30, 300)          0
 ropout1D)

 conv1d (Conv1D)             (None, 26, 64)            96064

 bidirectional (Bidirectiona  (None, 128)              66048
 l)

 dense (Dense)               (None, 512)               66048

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 512)               262656

 dense_2 (Dense)             (None, 1)                 513

=================================================================
Total params: 14,541,529
Trainable params: 491,329
Non-trainable params: 14,050,200
_________________________________________________________________
```

```
## Early Stopping object:
early_stopping = EarlyStopping(monitor='val_loss',patience=5,mode="auto")

## Model Checkpoint Object:
checkpoint_path = 'model_checkpoints/'
checkpoint = ModelCheckpoint(
    filepath=checkpoint_path,
    save_freq='epoch',
    save_weights_only=True,
    verbose=1
```

```
)

## Automatic learning rate update:

#monitor='val_loss' to use validation loss as performance measure to reduce the learning rate.
#patience=2 means the learning rate is reduced as soon as 2 epochs with no improvement.
#min_delta=0.001 means the validation loss has to improve by at least 0.001 for it to count as an improvement.
#factor=0.2 means the new learning rate will be reduced as new_lr = lr * factor

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.1,
    patience=5,
    min_lr=0.001,
    verbose=1
)

## Learning Rate Scheduler:
def lr_decay(epoch, lr):
    if epoch != 0 and epoch % 5 == 0:
        return lr * 0.2
    return lr

lrs=LearningRateScheduler(lr_decay, verbose=1)


print("Training on GPU...") if tf.test.is_gpu_available() else print("Training on CPU...")
```

```
WARNING:tensorflow:From <ipython-input-61-d4a4e34b7f9f>:1: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
Training on CPU...
```

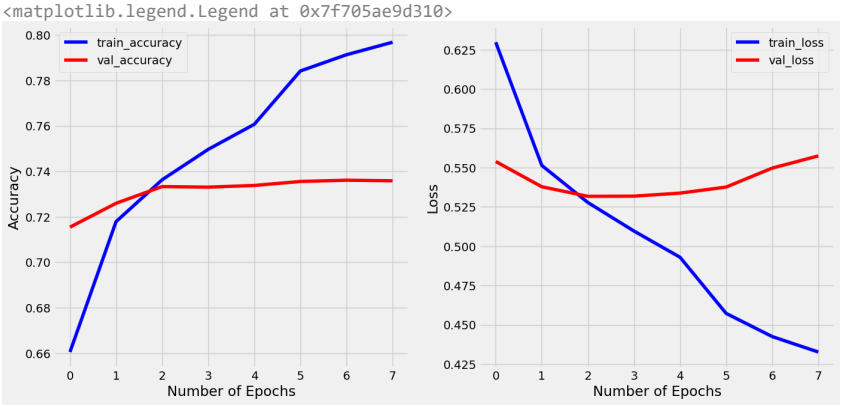
```
batch_size = 1000
epochs = 20
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
                    validation_data=(x_test, y_test), callbacks=[early_stopping,lrs])
```

```
Epoch 1: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 1/20
80/80 [==============================] - 125s 1s/step - loss: 0.6298 - accuracy: 0.6605 - val_loss: 0.5539 - val_accuracy: 0.7154 - lr:

Epoch 2: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 2/20
80/80 [==============================] - 115s 1s/step - loss: 0.5514 - accuracy: 0.7179 - val_loss: 0.5378 - val_accuracy: 0.7260 - lr:

Epoch 3: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 3/20
80/80 [==============================] - 99s 1s/step - loss: 0.5278 - accuracy: 0.7363 - val_loss: 0.5317 - val_accuracy: 0.7333 - lr:

Epoch 4: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 4/20
80/80 [==============================] - 101s 1s/step - loss: 0.5096 - accuracy: 0.7497 - val_loss: 0.5318 - val_accuracy: 0.7330 - lr:

Epoch 5: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 5/20
80/80 [==============================] - 98s 1s/step - loss: 0.4930 - accuracy: 0.7607 - val_loss: 0.5338 - val_accuracy: 0.7338 - lr:

Epoch 6: LearningRateScheduler setting learning rate to 0.0019999999552965165.
Epoch 6/20
80/80 [==============================] - 107s 1s/step - loss: 0.4572 - accuracy: 0.7842 - val_loss: 0.5376 - val_accuracy: 0.7355 - lr:

Epoch 7: LearningRateScheduler setting learning rate to 0.0019999999862164259.
Epoch 7/20
80/80 [==============================] - 111s 1s/step - loss: 0.4425 - accuracy: 0.7913 - val_loss: 0.5497 - val_accuracy: 0.7361 - lr:

Epoch 8: LearningRateScheduler setting learning rate to 0.0019999999862164259.
Epoch 8/20
80/80 [==============================] - 131s 2s/step - loss: 0.4327 - accuracy: 0.7968 - val_loss: 0.5574 - val_accuracy: 0.7358 - lr:
```

```
fig,ax=plt.subplots(1,2,figsize=(15,7))
ax[0].plot(history.history['accuracy'],c= 'b',label="train_accuracy")
ax[0].plot(history.history['val_accuracy'],c= 'r',label="val_accuracy")
ax[0].set_xlabel("Number of Epochs")
ax[0].set_ylabel("Accuracy")
ax[0].legend()
```

```
ax[1].plot(history.history['loss'],c= 'b',label="train_loss")
ax[1].plot(history.history['val_loss'],c= 'r',label="val_loss")
ax[1].set_xlabel("Number of Epochs")
ax[1].set_ylabel("Loss")
ax[1].legend()
```

```
<matplotlib.legend.Legend at 0x7f705ae9d310>
```



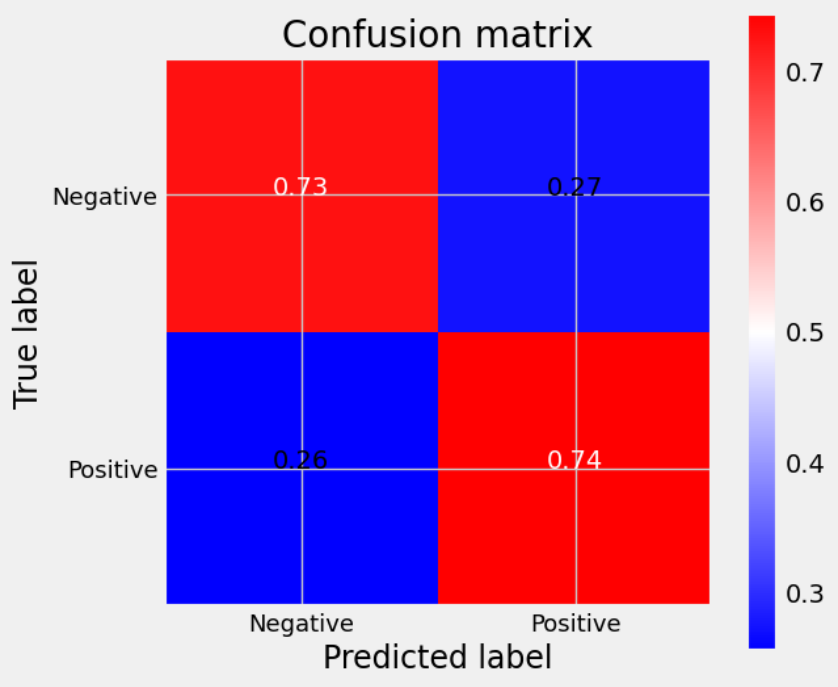

```
def decode_sentiment(score):
    return "Positive" if score>0.5 else "Negative"


scores = model.predict(x_test, verbose=1, batch_size=1000)
y_pred_1d = [decode_sentiment(score) for score in scores]
```

```
 20/20 [==============================] - 7s 341ms/step
```

```
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.bwr):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """

    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=20)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, fontsize=13)
    plt.yticks(tick_marks, classes, fontsize=13)

    fmt = '.2f'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
```

```
    plt.ylabel('True label', fontsize=17)
    plt.xlabel('Predicted label', fontsize=17)
```

```
cnf_matrix = confusion_matrix(test_data.sentiment.to_list(), y_pred_1d)
plt.figure(figsize=(6,6))
plot_confusion_matrix(cnf_matrix, classes=test_data.sentiment.unique(), title="Confusion matrix")
plt.show()
```




```
print(classification_report(list(test_data.sentiment), y_pred_1d))
```

```
                  precision    recall  f1-score   support

        Negative       0.74      0.73      0.73     10024
        Positive       0.73      0.74      0.74      9976

        accuracy                           0.74     20000
       macro avg       0.74      0.74      0.74     20000
    weighted avg       0.74      0.74      0.74     20000
```

More stuffs

```
new_df = pd.read_csv("archive/training.1600000.processed.noemoticon.csv",encoding='latin-1')
new_df.head()
```

	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D
0	0	1467810672	Mon Apr 06 22:19:49 PDT	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...

```
DATASET_COLUMNS = ["target", "ids", "date", "flag", "user", "TweetText"]
new_df.columns = DATASET_COLUMNS
```

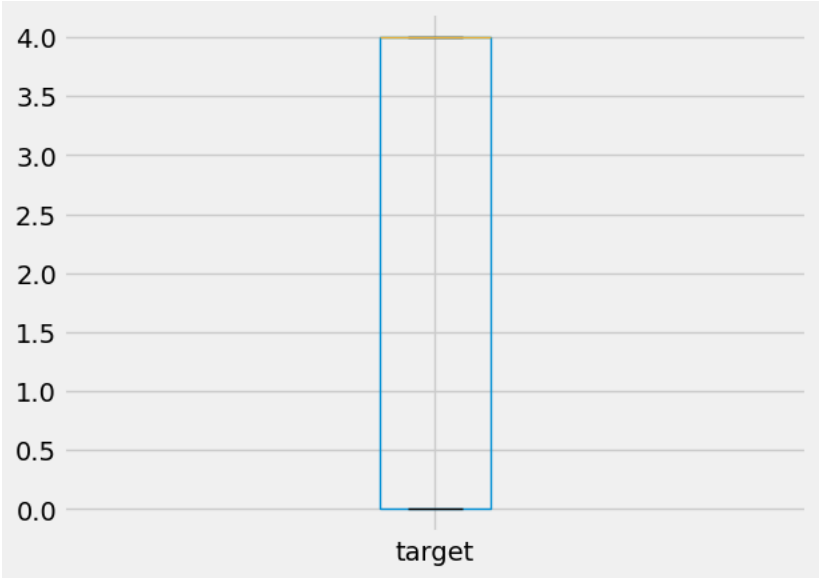
```
# lets ensure the 'date' column is in date format
new_df['date'] = pd.to_datetime(new_df['date'])
```

```
new_df.describe(include='all')
```

	target	ids	date	flag	user	TweetText
count	1.599999e+06	1.599999e+06	1599999	1599999	1599999	1599999
unique	NaN	NaN	774362	1	659775	1581465
top	NaN	NaN	2009-06-15 12:53:14	NO_QUERY	lost_dog	isPlayer Has Died! Sorry
freq	NaN	NaN	20	1599999	549	210
first	NaN	NaN	2009-04-06 22:19:49	NaN	NaN	NaN
last	NaN	NaN	2009-06-25 10:28:31	NaN	NaN	NaN
mean	2.000001e+00	1.998818e+09	NaN	NaN	NaN	NaN
std	2.000001e+00	1.935757e+08	NaN	NaN	NaN	NaN
min	0.000000e+00	1.467811e+09	NaN	NaN	NaN	NaN
25%	0.000000e+00	1.956916e+09	NaN	NaN	NaN	NaN
50%	4.000000e+00	2.002102e+09	NaN	NaN	NaN	NaN

```
import copy
data_ = copy.deepcopy(new_df)

positif_data = data_[data_.target==4].iloc[:80000,:]
negative_data = data_[data_.target==0].iloc[:80000,:]

sub_data = pd.concat([positif_data,negative_data],axis=0)


new_df.boxplot(column='target')
```

```
<Axes: >
```




```
data_target=new_df.groupby('target')


new_df['target'].value_counts()
```

```
4    800000
0    799999
Name: target, dtype: int64
```

```
new_df.head()
```

	target	ids	date	flag	user	TweetText
0	0	1467810672	2009-04-06 22:19:49	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by

```
data_ = {'target': new_df['target'], 'date': new_df['date']}
df1 = pd.DataFrame(data_)
df1.head()
```

	target	date
0	0	2009-04-06 22:19:49
1	0	2009-04-06 22:19:53
2	0	2009-04-06 22:19:57
3	0	2009-04-06 22:19:57
4	0	2009-04-06 22:20:00

```
hour = [ new_df['date'][i].hour for i in range(len(new_df['date'])) ]
new_df['hour'] = hour
new_df.head()
```

	target	ids	date	flag	user	TweetText	hour
0	0	1467810672	2009-04-06 22:19:49	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...	22
1	0	1467810917	2009-04-06 22:19:53	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...	22
2	0	1467811184	2009-04-06	NO_QUERY	ElleCTF	my whole body feels itchy and like its on	22

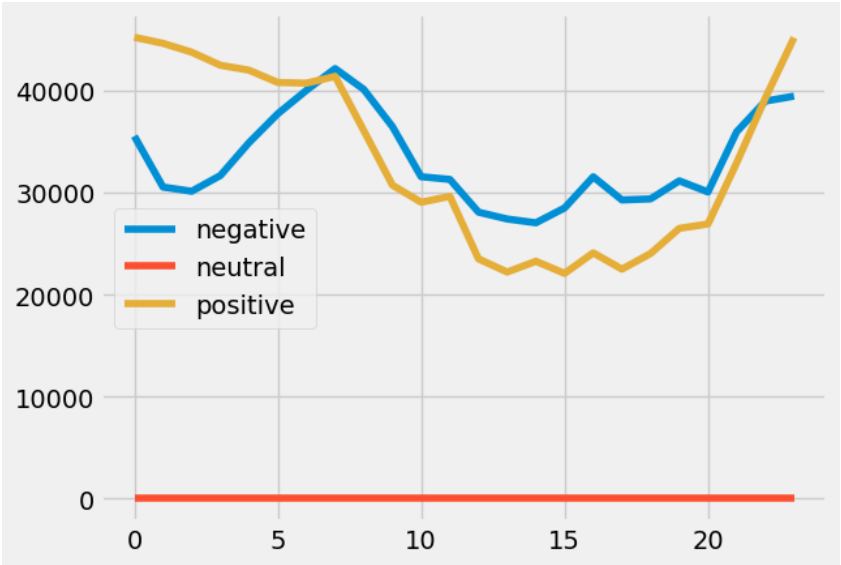
```
hour_data = {'0': [0]*24, '2': [0]*24, '4': [0]*24}
for i in range(len(new_df['hour'])):
    target = str(new_df['target'][i])
    hour = int(new_df['hour'][i])
    hour_data[target][hour] += 1
```

```
hour_data = [hour_data['0'], hour_data['2'], hour_data['4']]
# Transpose
hour_data = list(map(list,zip(*hour_data)))
```

```
new_df1 = pd.DataFrame(hour_data,index = [i for i in range(24)],columns=['negative', 'neutral', 'positive'])
```

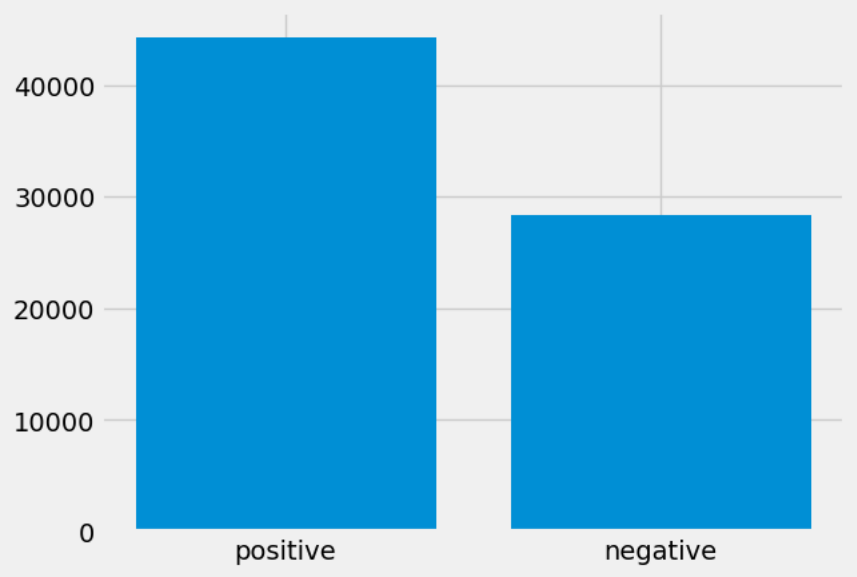
```
new_df1.plot()
```

```
<Axes: >
```



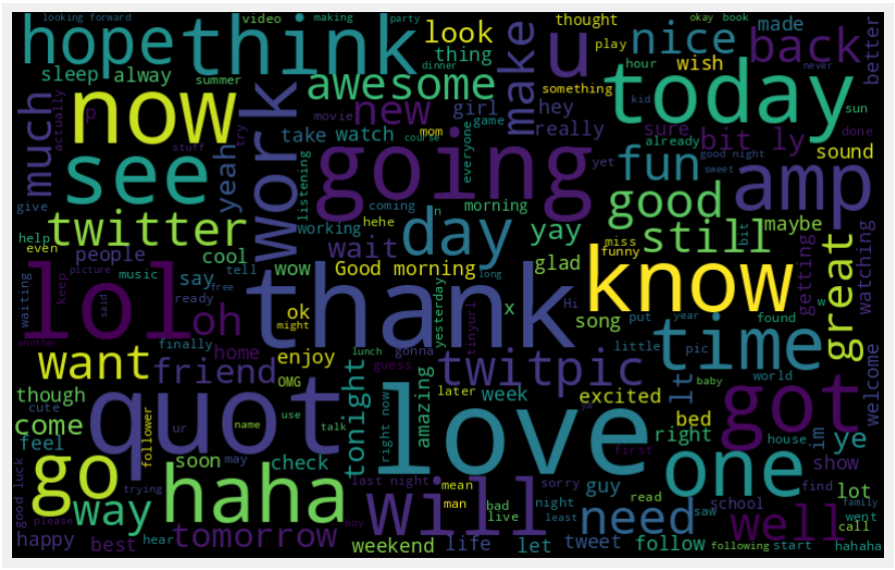
```
positive_at_count = 0
negative_at_count = 0
TweetTextList = list(sub_data['TweetText'])
targetList = list(sub_data['target'])
for i in range(len(sub_data['TweetText'])):
    if TweetTextList[i].find('@') != -1:
        if targetList[i] == 4:
            positive_at_count += 1
        else:
            negative_at_count += 1
at_counts = [positive_at_count, negative_at_count]


names = ['positive', 'negative']
values = [positive_at_count, negative_at_count]
plt.bar(names, values)
```

```
<BarContainer object of 2 artists>
```

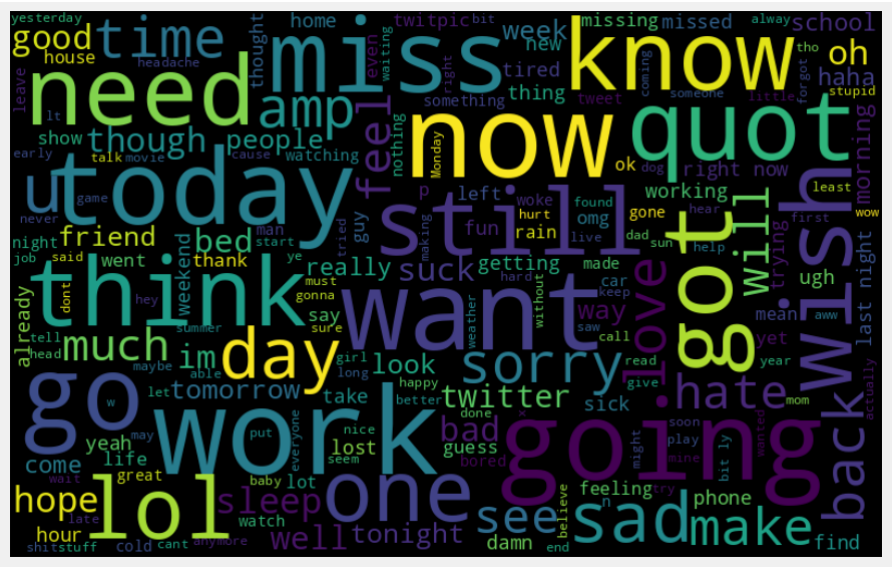



```
import copy
newdata = copy.deepcopy(sub_data)
newdata.drop(['ids','date','flag','user'],axis = 1,inplace = True)


all_words = ' '.join([text for text in newdata['TweetText']])
wordcloud = WordCloud(width=800,height=500,random_state=21,max_font_size=110).generate(all_words)
plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```


```
positive_words = ' '.join([text for text in new_df['TweetText'][new_df['target']==4]])
wordcloud = WordCloud(width=800,height=500,random_state=21,max_font_size=110).generate(positive_words)
plt.figure(figsize=(10,7))
plt.imshow(wordcloud,interpolation="bilinear")
plt.axis('off')
plt.show()
```




```
negative_words = ' '.join([text for text in new_df['TweetText'][new_df['target']==0]])
wordcloud = WordCloud(width=800,height=500,random_state=21,max_font_size=110).generate(negative_words)
plt.figure(figsize=(10,7))
plt.imshow(wordcloud,interpolation="bilinear")
plt.axis('off')
plt.show()
```



✓ 49s completed at 12:50 PM