IMPLEMENTATION CODE

Twitter Streaming Code:

```
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream

consumer_key=""
consumer_secret=" "
access_token=" "
access_token_secret=""
class TwitterStreamer():

    def __init__(self):
        pass

    def stream_tweets(self, fetched_tweets_filename, hash_tag_list):

        listener = StdOutListener(fetched_tweets_filename)
        auth = OAuthHandler(consumer_key,consumer_secret)
        auth.set_access_token(access_token,access_token_secret)
        stream = Stream(auth, listener)
        stream.filter(track=hash_tag_list,languages=['en'])


class StdOutListener(StreamListener):
    def __init__(self, fetched_tweets_filename):
        self.fetched_tweets_filename = fetched_tweets_filename

    def on_data(self, data):
        try:

            with open(self.fetched_tweets_filename, 'a') as tf:
                tf.write(data)
            return True
        except BaseException as e:
            print("Error on_data %s" % str(e))
        return True


    def on_error(self, status):
         if status == 420:
            return False
         print(status)
```

```
if __name__ == '__main__':


    hash_tag_list = ['#covid19', '#longcovid','#coronavirus','#stayhome','#socialdistancing','#covid-19','#covid2019','#coronavirusoutbreak', '#sarscov2', '#virus', '#covidisnotover', '#covidvaccines','#vaccinated', '#longcovid', '#omicron', '#cases', '#covid', '#pandemic', '#coronaviruspandemic', '#mask', '#deltacron', '#covidiots']
    fetched_tweets_filename = "final.json"

    twitter_streamer = TwitterStreamer()
    twitter_streamer.stream_tweets(fetched_tweets_filename, hash_tag_list)
```

Time Distribution Graph Code:

```
import numpy as np
from scipy.interpolate import make_interp_spline
import matplotlib.pyplot as plt




y = np.array([0,31562,40960,27205,32442,23713,35611,34859,38756,0])

x= [' Mar 24 23:30','Mar 25 11:30','Mar 25 23:30','Mar 26 11:30','Mar 26 23:30','Mar 27 12:30','Mar 28 00:30','Mar 28 12:30','Mar 29 00:30','Mar 29 12:30']

xf=np.arange((len(x)))
X_Y_Spline = make_interp_spline(xf, y)

# Returns evenly spaced numbers
# over a specified interval.
X_ = np.linspace(xf.min(), xf.max(), 500)
Y_ = X_Y_Spline(X_)

# plotting the points

plt.plot(X_,Y_, color= "red",linestyle="solid")
plt.gca().yaxis.set_major_formatter(plt.matplotlib.ticker.StrMethodFormatter('{x:,.0f}'))
plt.xticks(xf,x,rotation=90)

plt.xlabel('Time (GMT)')
# naming the y axis
plt.ylabel('Number of Tweets')
```

```
#plt.title('Distribution of Tweets over Retreival')

plt.grid()
plt.figure(figsize=(20,15))
#plt.savefig('plot.png', dpi=300)
plt.show()
```

Histogram Code:

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
from matplotlib.ticker import PercentFormatter
x=['Surprise','Anger','Sadness','Fear','Joy','Love']
y=[572,2159,4666,1937,5362,1304]
mpl.rcParams['axes.spines.right'] = False
mpl.rcParams['axes.spines.top'] = False
plt.rcParams.update({'font.family':'Times'})
plt.rcParams.update({'font.size': 10})
barlist=plt.bar(x, y)
plt.gca().yaxis.set_major_formatter(plt.matplotlib.ticker.StrMethodFormatter('{x:,.0f}'))
#plt.gca().yaxis.set_major_formatter(PercentFormatter(1,0))
barlist[0].set_color('yellow')
barlist[1].set_color('r')
barlist[2].set_color('b')
barlist[3].set_color('orange')
barlist[4].set_color('g')
barlist[4].set_color('purple')
plt.savefig('train.eps',format='eps',dpi=1200,facecolor='w',bbox_inches='tight')

plt.show()
```

Polarity Count code:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
sid_obj= SentimentIntensityAnalyzer()

def pos(tweet):
    return sid_obj.polarity_scores(tweet)['pos']
def neu(tweet):
    return sid_obj.polarity_scores(tweet)['neu']
def neg(tweet):
    return sid_obj.polarity_scores(tweet)['neg']
def comp(tweet):
    return sid_obj.polarity_scores(tweet)['compound']
vader_df=pd.DataFrame()
vader_df['Tweet']=df['Tweet Text']
vader_df['Positive']=vader_df['Tweet'].apply(pos)
vader_df['Neutral']=vader_df['Tweet'].apply(neu)
vader_df['Negative']=vader_df['Tweet'].apply(neg)
vader_df['Compound']=vader_df['Tweet'].apply(comp)
vader_df
```

```
pos=[]
neg=[]
neu=[]
for i in df['Polarity']:
    if i>(0.2):
        pos.append(i)
    elif i<(-0.2):
        neg.append(i)
    else:
        neu.append(i)
posa=[abs(number) for number in pos]
nega=[abs(number) for number in neg]
neua=[abs(number) for number in neu]
print(sum(posa))
print(sum(nega))
print(sum(neua))
print(len(posa))
print(len(nega))
print(len(neua))
```

Text2Emotion:

```
def text_emotion(tweet):
    return te.get_emotion(tweet)
emotion=pd.DataFrame()
emotion['Text']=df['Text']
emotion['Text2Emotion']=emotion['Text'].apply(text_emotion)
emotion
```

```
text2emo=pd.DataFrame()
text2emo['Text']=emotion['Text']
happy=[]
angry=[]
surprise=[]
sad=[]
fear=[]
for i in emotion['Text2Emotion']:
    happy.append(i['Happy'])
    angry.append(i['Angry'])
    surprise.append(i['Surprise'])
    sad.append(i['Sad'])
    fear.append(i['Fear'])

text2emo['Happy']=happy
text2emo['Angry']=angry
text2emo['Surprise']=surprise
text2emo['Sad']=sad
text2emo['Fear']=fear
text2emo
```

LSTM Model:

```
EMBEDDING_DIM = 100
class_num = 6
 model = Sequential()
model.add(Embedding(input_dim = num_words,
output_dim = EMBEDDING_DIM,
input_length= X_train_pad.shape[1],
 weights = [gensim_weight_matrix],trainable = False))
model.add(Dropout(0.2))
model.add(Bidirectional(CuDNNLSTM(100,return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(CuDNNLSTM(200,return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(CuDNNLSTM(100,return_sequences=False)))
model.add(Dense(class_num, activation = 'softmax')) model.compile(loss =
'categorical_crossentropy', optimizer = 'adam',metrics = 'accuracy')
```