

**School of Computer Science and Engineering**

**J Component report**

**Programme : M.Tech Integrated(CSE)**

**Course Title: Natural Language Processing**

**Course Code: SWE1017**

**Slot: B2**

**Title: Suicidal Text Analysis Using NLP**

**Team Members: Sayantan Nandy | 19MIA1049**

**Gitansh Saharan | 19MIA60**

**Mohit Madhukar More | 19MIA1005**

**Faculty: R. SRIDHAR**



**School of Computer Science and Engineering**

**DECLARATION**

I hereby declare that the project entitled **“Suicidal Text Analysis Using NLP”** submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai Campus, Chennai 600127 in partial fulfilment of the requirements for the award of the degree of M.Tech (Integrated) Business Analytics **– Computer Science and Engineering** is a record of bonafide work carried out by me**.** I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

Signature

**Mohit More (19MIA1005)**

**Gitansh Saharan | 19MIA60**

**Sayantan Nandy (19MIA1049)**

**ABSTRACT**

It is estimated that each year many people, most of whom are teenagers and young adults die by suicide worldwide. Suicide receives special attention with many countries developing national strategies for prevention. It is found that, social media is one of the most powerful tool from where we can analyze the text and estimate the chances of suicidal thoughts. Using nlp we can analyze twitter and reddit texts monitor the actions of that person.

The most difficult part to prevent suicide is to detect and understand the complex risk factors and warning signs that may lead to suicide. This can be achieved by identifying the sudden changes in a user’s behavior automatically. [Natural language processing](https://www.sciencedirect.com/topics/computer-science/natural-language-processing) techniques can be used to collect behavioral and textual features from social media interactions and these features can be passed to a specially designed framework to [detect anomalies](https://www.sciencedirect.com/topics/computer-science/detect-anomaly) in human interactions that are indicators of suicidal intentions. We can achieve fast detection of suicidal ideation using deep learning and/or [machine learning](https://www.sciencedirect.com/topics/computer-science/machine-learning) based [classification](https://www.sciencedirect.com/topics/computer-science/classification) approaches.

*Keywords:*

*Machine learning*

*Deep learning*

*Natural language processing*

*Classification*

**INTRODUCTION**

The rise of social media and online communities creates safe and anonymous spaces for individuals to share their thoughts about their mental health and express their feelings and sufferings in online communities. To prevent suicide, it is necessary to detect suicide-related posts and user's suicide ideation in cyberspace by natural language processing methods. We focused on the online community called Reddit and the social networking website Twitter, and classify user's posts with potential suicide and without suicidal risk through text features processing, machine learning, and deep learning based methods.

**DATA PREPROCESSING**

- Perform text cleaning and will remove some corpus-specific stopwords. And will plot word cloud to visualize the frequently occurring words in a corpus.

- Perform vectorization using Both Bag of Words and TFIDF Vectorizer.

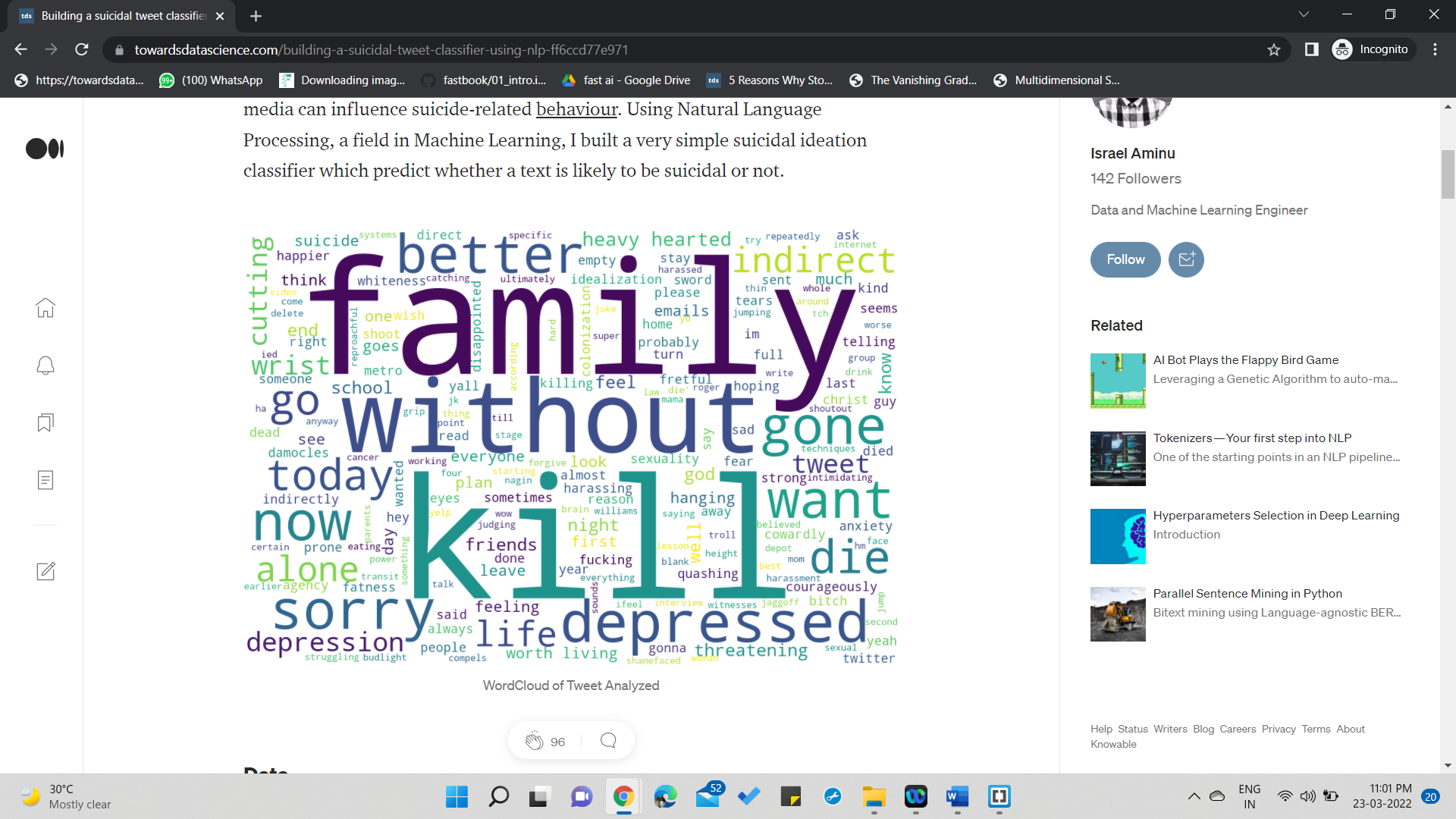
- Used grid search cv to find the best parameters to train the model using Random Forest Classifier and archived an accuracy of 96%.

**DATASET**

Collected two sets of data from Reddit and Twitter. The Reddit data set includes (2958) suicidal ideation samples and a number of non-suicide texts (5381). The Twitter dataset has a total (3000) tweets with suicidal ideation.

Reddit Data was scraped from subreddits like 'suicide watch', 'depression', 'anxiety' etc. Twitter data was collected by querying keywords like 'end my life', 'die' etc.

**Twitter**



**Application**

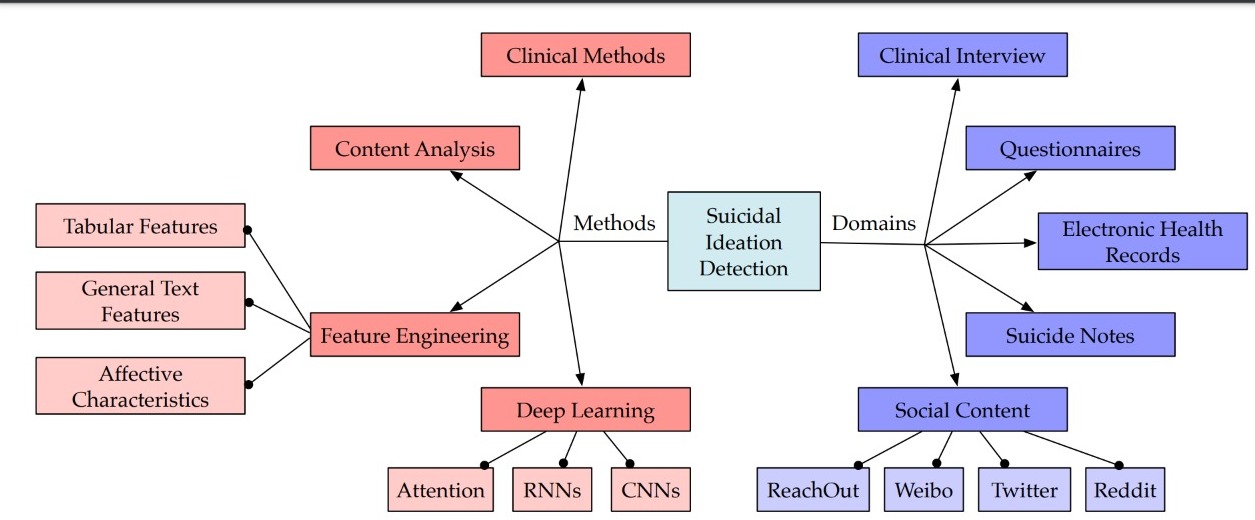
Text analysis is having a wide range of application from analyzing patients mental condition from EHR to mental health to suicidal chances.

Using EHR analysis we can predict the patients physical and mental condition. The increasing volume of electronic health records (EHRs) has paved the way for machine learning techniques for suicide attempter prediction. Patient records include demographical information and diagnosis-related history like admissions and emergency visits

Mental health, using a special set of questionnaire , we can predict the mental health of that person . The questionnaire will deal with certain questions through which we can make a guess of his / hear health\

Out of all the application we are aiming to develop an application which can predict whether the text is suicidal or not. As in the recent year we can observe a rise of social media application like whatsapp, twitter , facebook, reddit and it has been observed that people share their feeling on these platform and analyzing these texts can actually help to reduce the rate as we can predict it much before what that person is thinking.

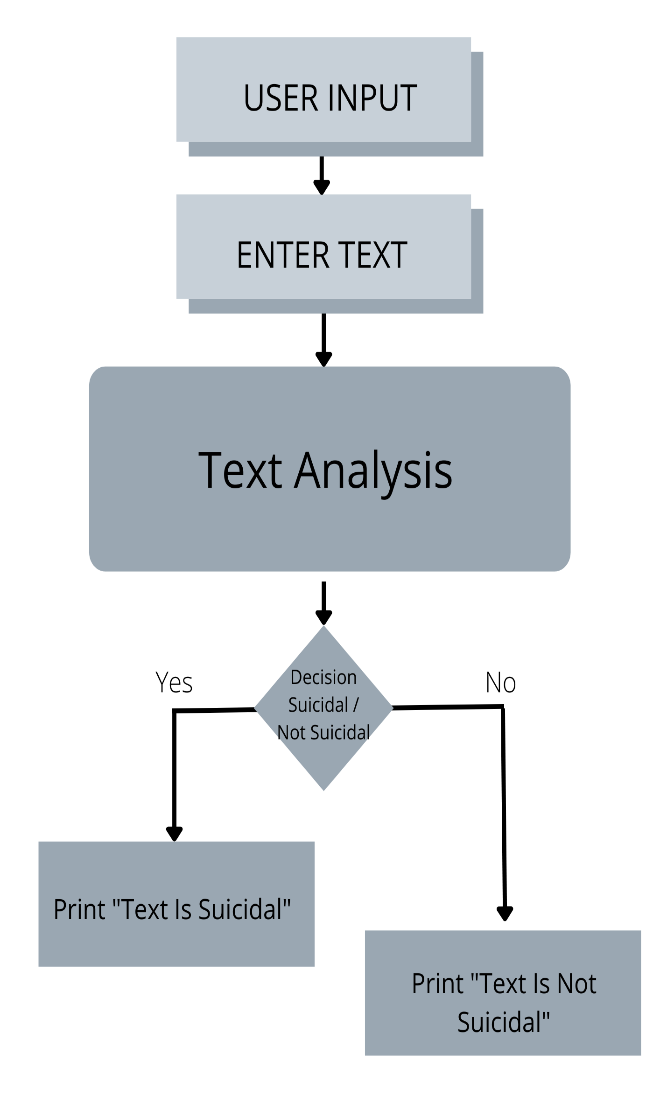
In order to develop this application we are building a model which is tokenizing each and every word and we are using glove method for word embedding. Later after preparing the text we have used BiLSTM to predict whether the text is suicidal or not.



The categorization of suicide ideation detection: methods and domains. The left part represents method

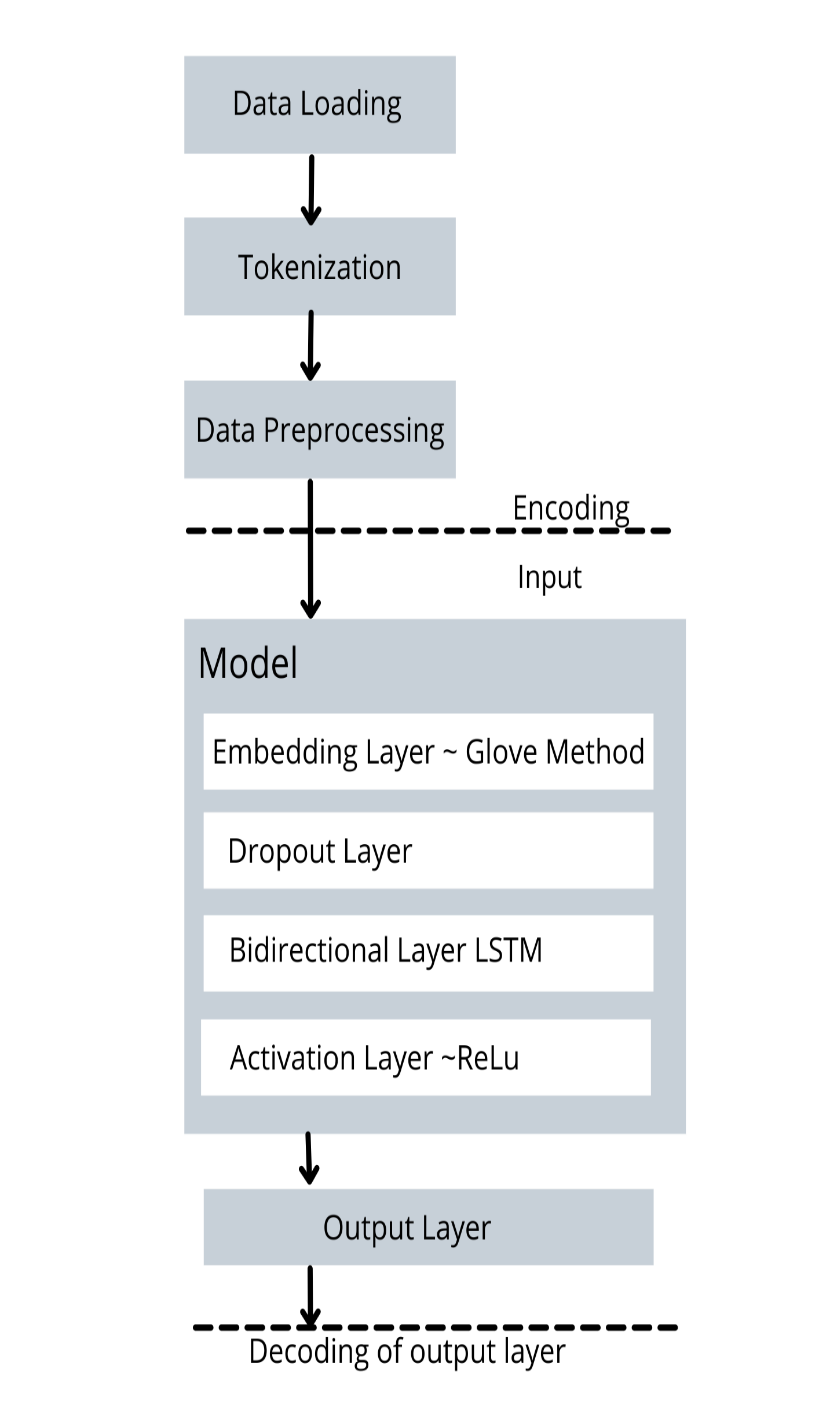
categorization, while the right part shows the categories of domains. The arrow and solid point indicate subcategories.

**Use Case Diagram**



1. We will be entering the text
2. Our model will tokenize it and predict the suicidal rate
3. At last we can observe whether it is suicidal or not.

**Model Diagram**



**Process :**

After loading the data into the workspace, we have applied various pre preprocessing techniques such as tokenization and word embedding on the dataset. Glove Twitter Data was used for word embedding.

Extracted text from the given  
text in the dataset and saved it into the tokenThese tokens are then converted to sequences using Tweet Tokenizer. We have created x and y for text data and sequences. X and Y were then split to train and validation set.

Different processes that we have used are explained below.

**BiLstm**

The first layer was taken to be the input layer. The second layer was taken to be the embedding layer during the making of the model. We have taken the dropout layer as 0.1, while the bi directional layer

was the fourth layer.In bidirectional, our input flows in two directions, making a BI-LSTM different from the regular LSTM. With the regular LSTM, we can make input flow in one direction, either backwards or forward. However, in bi-directional,

we can make the input flow in both directions to pre- serve the future and the past information. A bidirectional LSTM (BiLSTM) layer learns bidirectional

long-term dependencies between time steps of time series or sequence data. These dependencies can be useful when you want the network to learn from the complete time series at each time step

**Tokenization**

As tokens are the building blocks of Natural Language, the most common way of processing the raw text happens at the token level. Tokenization is the foremost step while modeling text data. Tokenization is performed on the corpus to obtain tokens. Word Tokenization is the most commonly used tokenization algorithm. It splits a piece of text into individual words based on a certain delimiter. Depending upon delimiters, different word-level tokens are formed. [Pretrained Word Embeddings](https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/?utm_source=blog&utm_medium=what-is-tokenization-nlp) such as Word2Vec and GloVe comes under word tokenization. One of the major issues with word tokens is dealing with**Out Of Vocabulary (OOV) words**. OOV words refer to the new words which are encountered at testing. These new words do not exist in the vocabulary. Hence, these methods fail in handling OOV words.

**Word Embedding**

Pretrained Word Embeddings are the embeddings learned in one task that are used for solving another similar task. These embeddings are trained on large datasets, saved, and then used for solving other tasks. That’s why pretrained word embeddings are a form of **Transfer Learning.**

Pretrained word embeddings capture the semantic and syntactic meaning of a word as they are trained on large datasets. They are capable of boosting the performance of a [Natural Language Processing (NLP)](https://courses.analyticsvidhya.com/courses/natural-language-processing-nlp?utm_source=blog&utm_medium=pretrained-word-embeddings-nlp) model.

There are 2 major types of word embedding

1. Word2vec

Word2Vec is one of the most popular pretrained word embeddings developed by Google. Word2Vec is trained on the Google News dataset (about 100 billion words). It has several use cases such as [Recommendation Engines](https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/?utm_source=blog&utm_medium=pretrained-word-embeddings-nlp), Knowledge Discovery, and also applied in the different [Text Classification](https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/?utm_source=blog&utm_medium=pretrained-word-embeddings-nlp) problems.

1. GloVe

The basic idea behind the GloVe word embedding is to derive the relationship between the words from Global Statistics

One of the simplest ways is to look at the co-occurrence matrix. **A co-occurrence matrix tells us how often a particular pair of words occur together. Each value in a co-occurrence matrix is a count of a pair of words occurring together.**

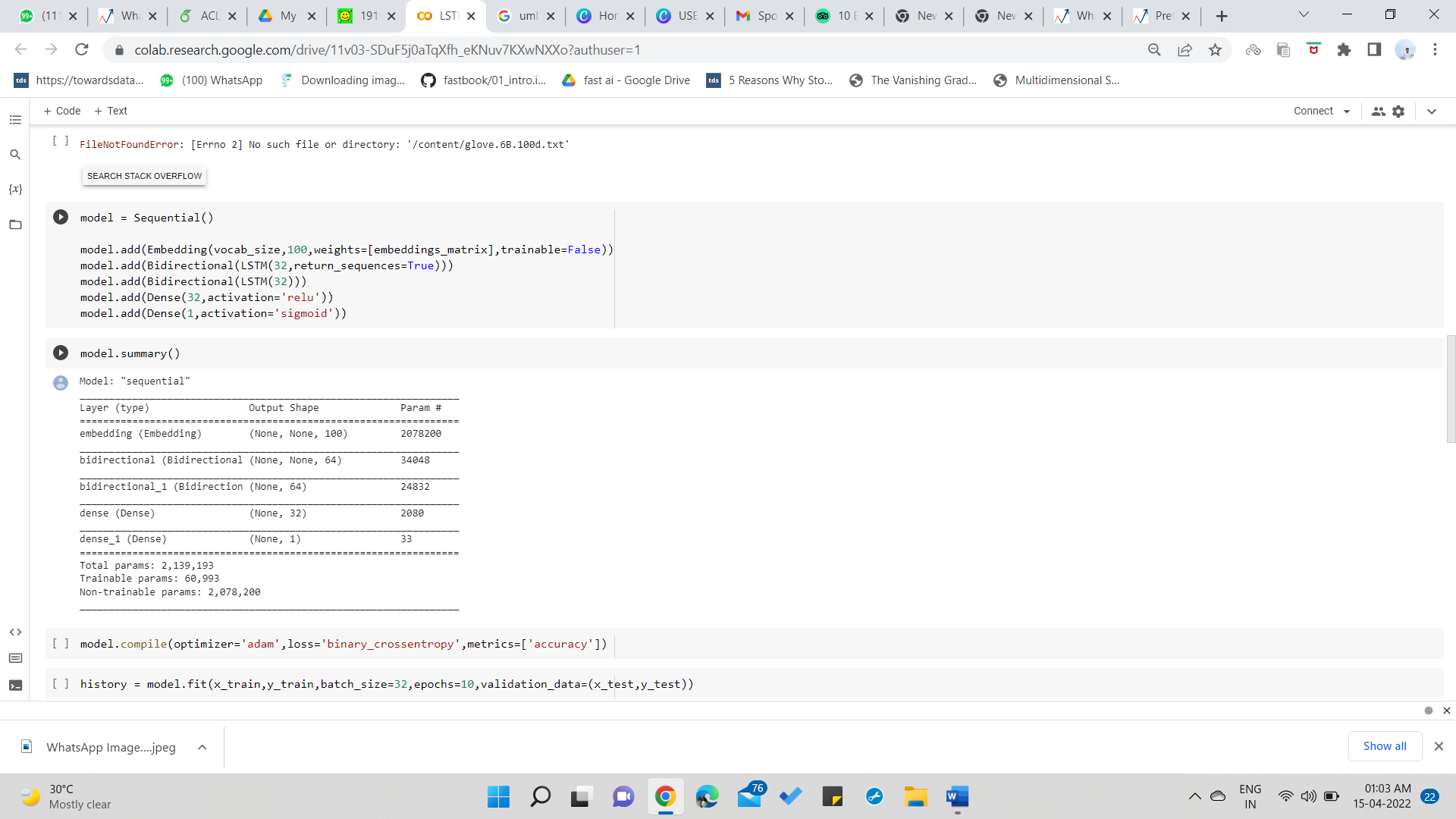
In this application we will be using GloVe method for wording embedding.

**Glove**

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Train-  
ing is performed on aggregated global word-word co-occurrence statistics from a corpus, and the re-  
sulting representations showcase interesting linear substructures of the word vector space.

**The model we prepared**

**Model :**



**Pseudocode**

**Model 1 : CNN**

a. First is the input layer where we will add the dataset of 512 \* 768 size.

b. Then we will apply Conv1D layer with filter size as 3 and kernel size as 2.

c. Then we applied dropout layer of 25 % dropout.

d. Flatten layer was applied followed by Dense layer with activation function as 'relu' and kernel initializer as 'he\_uniform'.

e. Then the output layer was applied with sigmoid function.

**Model 2: Bilstm**

a. First is the input layer where we will add the dataset of 512 \* 768 size.

b. The second layer was of Bidirectional LSTM with 25 % dropout and 20 % recurrent dropout.

c. Next layer was of MaxPolling Layer to reduce the size of input. The pool size of the MaxPolling Layer was 2.

d. Flatten layer was applied followed by Dense layer with activation function as 'relu' and kernel initializer as 'he\_uniform'.

e. Then the output layer was applied with sigmoid function.

**Working Code**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

import re

import nltk

from nltk.stem import WordNetLemmatizer

from nltk.corpus import stopwords

from nltk import FreqDist

from nltk import ngrams

import spacy

%matplotlib inline

wordnet = WordNetLemmatizer()

sp = spacy.load('en\_core\_web\_sm')

all\_stopwords = sp.Defaults.stop\_words

def clean\_text(text,stopwords):

    text = re.sub('[^a-zA-Z]', ' ',text)

    text = text.lower()

    text = text.split(' ')

    text = [wordnet.lemmatize(word) for word in text]

    text = [word for word in text if word not in stopwords]

    text = ' '.join(text)

    return text

def combine\_data(list1, list2):

    combined\_data = []

    for i in range(len(list1)):

        new\_str = list1[i] +' ' + list2[i]

        combined\_data.append(new\_str)

    return combined\_data

def generate\_frequency(text\_list):

    fdist =  FreqDist()

    for i in text\_list:

        words = i.split(' ')

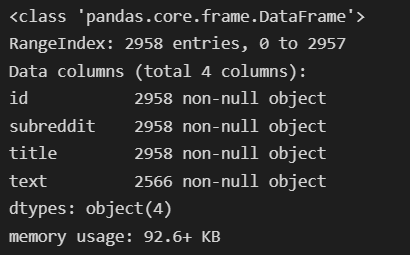
        words = [word for word in words if word != '']

        words = ngrams(words,1)

        for x in words:

            fdist[x[0]]+=1

    return fdist



new\_words = ["http", "www", "co", "u", "com", "t", "s", "m",

             "ve", "dy", "ll", 'n', 'r', 'b', "wa", "y", "don", "ha"]

for words in new\_words:

    all\_stopwords.add(words)

df['text'].fillna(value=' ',inplace=True)

df['title'] = df['title'].apply(clean\_text, stopwords = all\_stopwords)

df['text'] = df['text'].apply(clean\_text, stopwords = all\_stopwords)

df['cleaned'] = combine\_data(df['title'],df['text'])

df.to\_csv('./cleanedRedditSuicide.csv', index=False)

fdist = generate\_frequency(df['cleaned'])

top\_words = fdist.most\_common(n=150)

word\_dict = {}

for i in range(len(top\_words)):

    word\_dict[top\_words[i][0]] = top\_words[i][1]

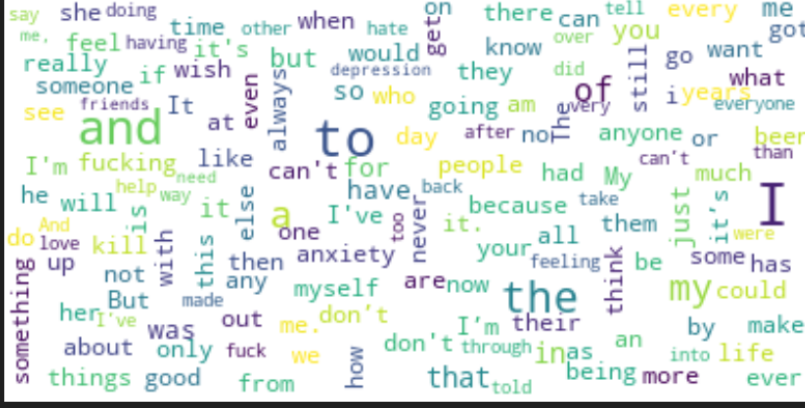
wordcloud = WordCloud(background\_color="white",max\_font\_size=30).generate\_from\_frequencies(word\_dict)

plt.figure(figsize = (14, 8))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.show()



df = pd.read\_csv('../Dataset/NoSuicideData.csv')

df['text'].fillna(value=' ',inplace=True)

df['title'] = df['title'].apply(clean\_text,stopwords = all\_stopwords)

df['text'] = df['text'].apply(clean\_text,stopwords = all\_stopwords)

df['cleaned'] = combine\_data(df['title'],df['text'])

df.to\_csv('./cleanedRedditNonSuicide.csv', index=False)

fdist = generate\_frequency(df['cleaned'])

top\_words = fdist.most\_common(n=150)

word\_dict = {}

for i in range(len(top\_words)):

    word\_dict[top\_words[i][0]] = top\_words[i][1]

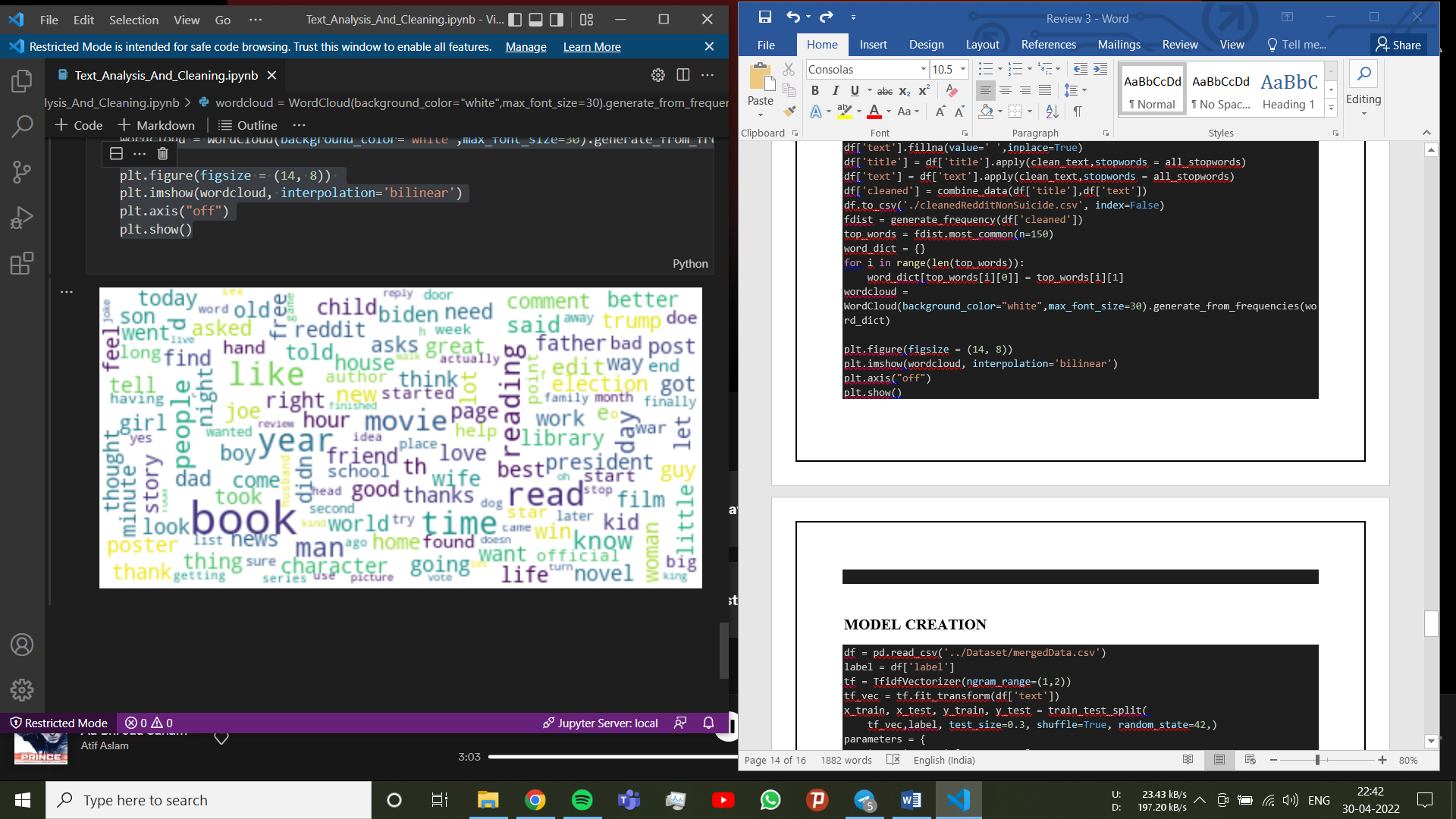
wordcloud = WordCloud(background\_color="white",max\_font\_size=30).generate\_from\_frequencies(word\_dict)

plt.figure(figsize = (14, 8))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.show()



**MODEL CREATION**

import pandas as pd

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split,cross\_val\_score

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report,accuracy\_score,confusion\_matrix

from sklearn.model\_selection import GridSearchCV

import pickle

df = pd.read\_csv('../Dataset/mergedData.csv')

label = df['label']

tf = TfidfVectorizer(ngram\_range=(1,2))

tf\_vec = tf.fit\_transform(df['text'])

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

    tf\_vec,label, test\_size=0.3, shuffle=True, random\_state=42,)

parameters = {

    'n\_estimators':[100,200,300],

    'max\_depth': [70, 80 , 90, 100]

}

model = RandomForestClassifier()

clf = GridSearchCV(model,param\_grid=parameters,scoring='accuracy',cv = 2)

clf.fit(tf\_vec,label)

clf.best\_params\_

model = RandomForestClassifier(n\_estimators=100,max\_depth=80)

model.fit(x\_train,y\_train)

cross\_val\_score(model,tf\_vec,label, scoring='accuracy')

pred = model.predict(x\_test)

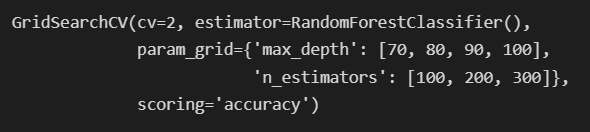
print(classification\_report(y\_test,pred))

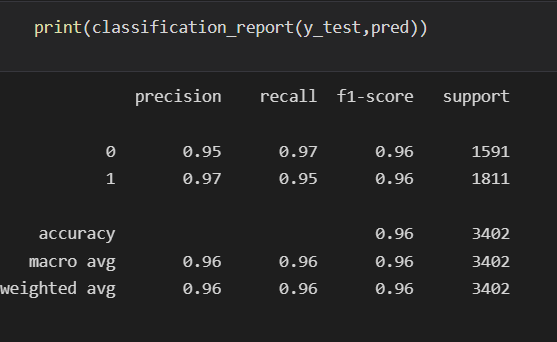
with open('tfidf\_tokenizer.pkl','wb') as f:

    pickle.dump(tf,f)

with open('random\_forest.pkl', 'wb') as f:

    pickle.dump(model,f)





**LSTM MODEL**

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Embedding, Dropout, Bidirectional

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

%matplotlib inline

df = pd.read\_csv('https://raw.githubusercontent.com/soumyajit4419/AI\_For\_Social\_Good/master/Dataset/mergedData.csv?token=AK7VCIERPG353P22MNQU4KDAJIQRQ')

text = df['text']

label = df['label']

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(text)

word\_index = tokenizer.word\_index

vocab\_size = len(word\_index) + 1

sequence = tokenizer.texts\_to\_sequences(text)

padded\_sequence = pad\_sequences(sequence,padding='post')

x\_train, x\_test, y\_train, y\_test = train\_test\_split(padded\_sequence,label,test\_size=0.3,shuffle=True,random\_state = 42)

embeddings\_index = {};

with open('/content/glove.6B.100d.txt') as f:

    for line in f:

        values = line.split();

        word = values[0];

        coefs = np.asarray(values[1:], dtype='float32');

        embeddings\_index[word] = coefs;

embeddings\_matrix = np.zeros((vocab\_size,100));

for word, i in word\_index.items():

    embedding\_vector = embeddings\_index.get(word);

    if embedding\_vector is not None:

        embeddings\_matrix[i] = embedding\_vector;

model = Sequential()

model.add(Embedding(vocab\_size,100,weights=[embeddings\_matrix],trainable=False))

model.add(Bidirectional(LSTM(32,return\_sequences=True)))

model.add(Bidirectional(LSTM(32)))

model.add(Dense(32,activation='relu'))

model.add(Dense(1,activation='sigmoid'))

model.summary()

model.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['accuracy'])

history = model.fit(x\_train,y\_train,batch\_size=32,epochs=10,validation\_data=(x\_test,y\_test))

fig,(ax1,ax2)=plt.subplots(nrows=1,ncols=2,figsize=(18,5))

ax1.plot(history.history['accuracy'],label='train\_accuracy')

ax1.plot(history.history['val\_accuracy'],label='test\_accuracy')

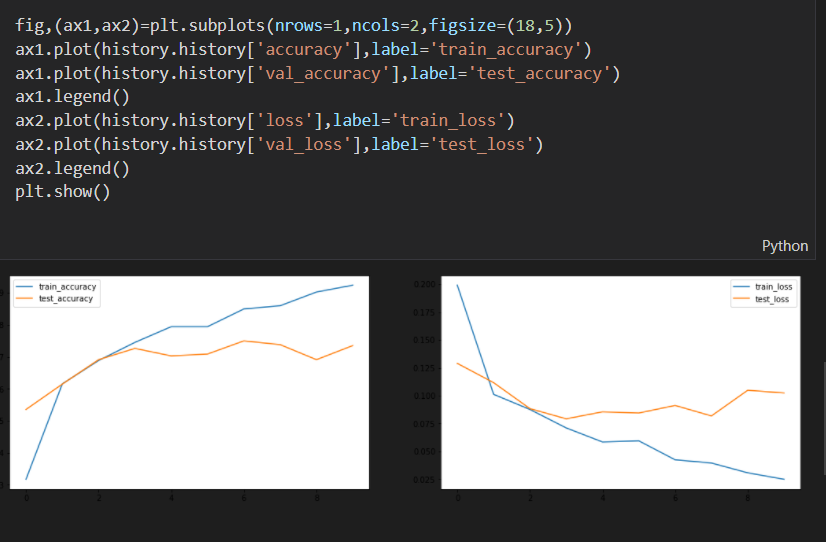
ax1.legend()

ax2.plot(history.history['loss'],label='train\_loss')

ax2.plot(history.history['val\_loss'],label='test\_loss')

ax2.legend()

plt.show()



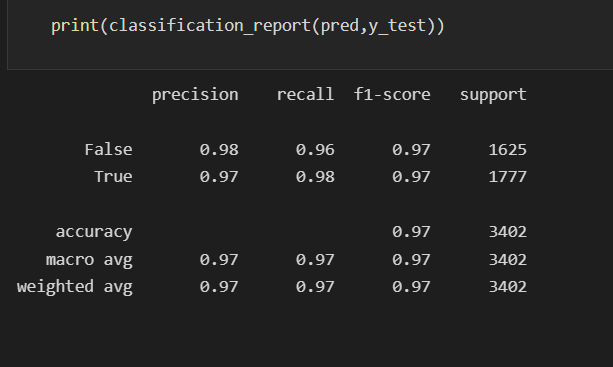
model.evaluate(x\_test,y\_test)

model.save('./lstm.h5')

pred = model.predict(x\_test)

pred = pred>0.5

print(classification\_report(pred,y\_test))



**Results:**

import preprocess\_text

import tensorflow as tf

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.models import load\_model

import pickle

tf\_idf = pickle.load(open('../Pretrained\_Models/tfidf\_tokenizer.pkl','rb'))

rf\_model = pickle.load(open('../Pretrained\_Models/random\_forest.pkl','rb'))

tokenizer = pickle.load(open('../Pretrained\_Models/tf\_tokenizer.pkl','rb'))

model = load\_model('../Pretrained\_Models/lstm.h5',compile=False)

def predict\_ml(text):

    new\_text = preprocess\_text.clean\_text(text)

    vec = tf\_idf.transform([new\_text])

    res = rf\_model.predict(vec)

    return res

def predict\_dl(text):

    new\_text = preprocess\_text.clean\_text(text)

    sequence = tokenizer.texts\_to\_sequences([new\_text])

    padded\_sequence = pad\_sequences(sequence,padding='post',)

    res = model.predict(padded\_sequence)

    return res[0]

text = ['i am watching a suicide movie', 'i dont want to live any more',

        'she commited suicide', 'Do you like to go out for a movie']

for t in text:

    res = predict\_dl(t)

    res = res > 0.5

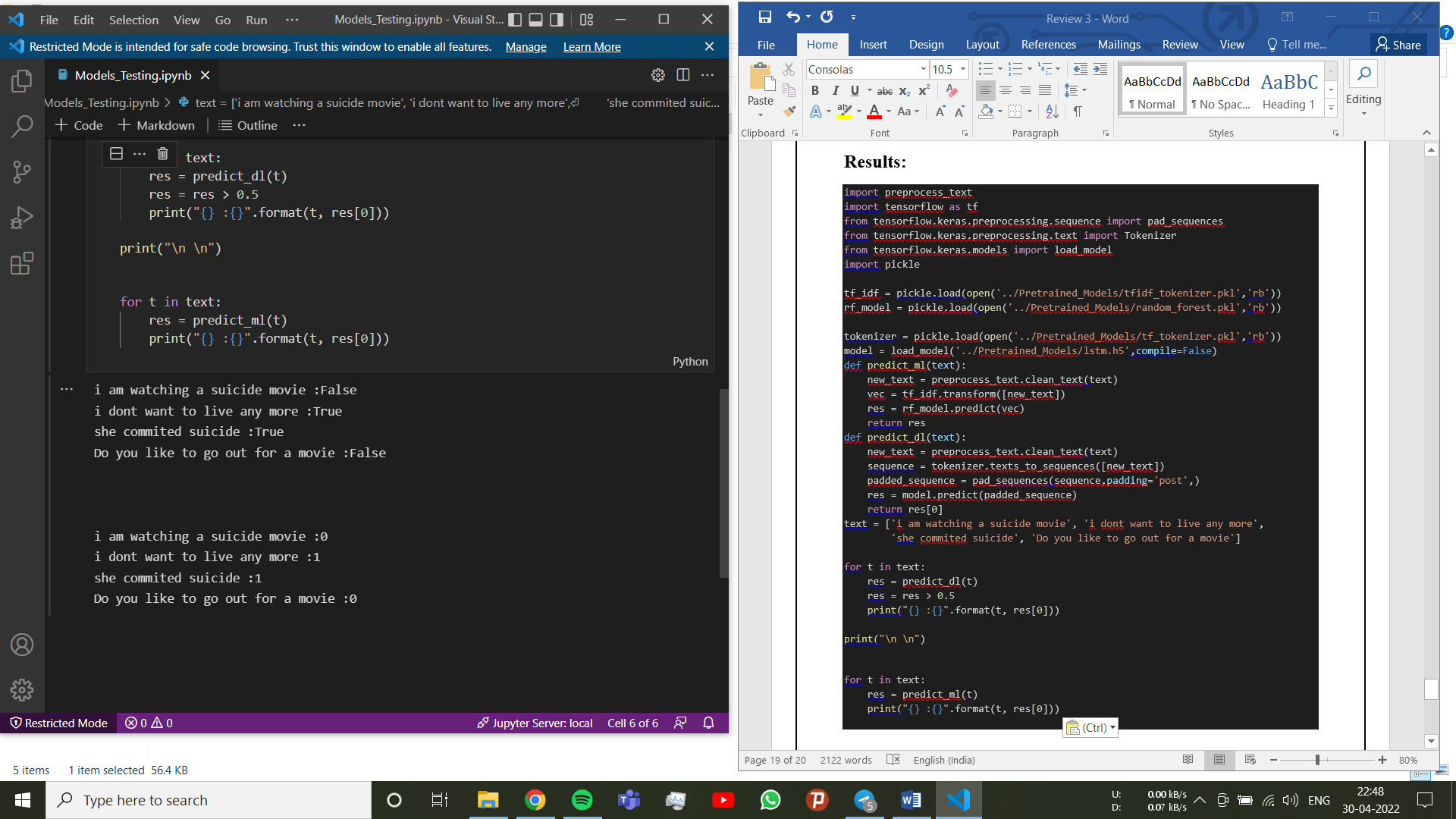
    print("{} :{}".format(t, res[0]))

print("\n \n")

for t in text:

    res = predict\_ml(t)

    print("{} :{}".format(t, res[0]))



We got the accuracy of 96% as f1-score for the Random forest model 97% and for the LSTM model we got the accuracy of 97% as f1-score. We applied the prepressing on the text data using Natural Language Tool Kit Library which is used for NLP prepressing and then used Machine Learning Models and Deep Learning Models.

**REFERENCES**

1. [*https://www.ijournalse.org/index.php/ESJ/article/view/34*](https://www.ijournalse.org/index.php/ESJ/article/view/34)
2. [*https://towardsdatascience.com/building-a-suicidal-tweet-classifier-using-nlp-ff6ccd77e971*](https://towardsdatascience.com/building-a-suicidal-tweet-classifier-using-nlp-ff6ccd77e971)
3. [*https://www.sciencedirect.com/science/article/pii/S1319157821003244*](https://www.sciencedirect.com/science/article/pii/S1319157821003244)
4. [*https://informatics.bmj.com/content/27/3/e100175*](https://informatics.bmj.com/content/27/3/e100175)