In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import neattext.functions as nfx
import plotly.express as plx
from sklearn.metrics import classification_report
import keras
from keras.layers import Embedding,Dense,LSTM,GlobalMaxPooling1D,Input
from keras.callbacks import EarlyStopping,ReduceLROnPlateau
from keras.models import Sequential
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from tqdm import tqdm
```

In [2]:
```
data= pd.read_csv(r"C:\Users\91937\Downloads\Suicide_Detection.csv (1).zip")
data.head()
```

Out[2]:

	Unnamed: 0	text	class
0	2	Ex Wife Threatening SuicideRecently I left my ...	suicide
1	3	Am I weird I don't get affected by compliments...	non-suicide
2	4	Finally 2020 is almost over... So I can never ...	non-suicide
3	8	i need helpjust help me im crying so hard	suicide
4	9	I’m so lostHello, my name is Adam (16) and I’v...	suicide

In [3]:
```
data['class'].value_counts()
```

Out[3]:
```
class
suicide        116037
non-suicide    116037
Name: count, dtype: int64
```

In [4]:
```
data['class'].value_counts().index.values
```

Out[4]:
```
array(['suicide', 'non-suicide'], dtype=object)
```

In [5]:
```
train_data,test_data=train_test_split(data,test_size=0.2,random_state=10)
```

In [6]:
```
train_data['class'].value_counts().index.values
```

Out[6]:
```
array(['suicide', 'non-suicide'], dtype=object)
```

In [7]:
```
# Import necessary libraries
import plotly.express as px
import pandas as pd

# Create a DataFrame from the value_counts
class_counts = train_data['class'].value_counts().reset_index()
class_counts.columns = ['class', 'count']

# Plot the bar chart
```

```
fig = px.bar(class_counts, x='class', y='count', color='class', title="Class Dis
fig.show()
```

In [8]:

```
def clean_text(text):
    text_length=[]
    cleaned_text=[]
    for sent in tqdm(text):
        sent=sent.lower()
        sent=nfx.remove_special_characters(sent)
        sent=nfx.remove_stopwords(sent)
        text_length.append(len(sent.split()))
        cleaned_text.append(sent)
    return cleaned_text,text_length
```

In [9]:

```
cleaned_train_text,train_text_length=clean_text(train_data.text)
cleaned_test_text,test_text_length=clean_text(test_data.text)
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 18
5659/185659 [00:29<00:00, 6258.22it/s]
100%|██████████████████████████████████████████████████████████████████████████████████|
46415/46415 [00:09<00:00, 4954.49it/s]
```

In [10]:

```
tokenizer=Tokenizer()
tokenizer.fit_on_texts(cleaned_train_text)
```

In [11]:

```
#cleaned_train_text
```

In [12]:
```
train_text_seq=tokenizer.texts_to_sequences(cleaned_train_text)
train_text_pad=pad_sequences(train_text_seq,maxlen=50)


test_text_seq=tokenizer.texts_to_sequences(cleaned_test_text)
test_text_pad=pad_sequences(test_text_seq,maxlen=50)
```

In [13]:
```
train_text_pad
```

Out[13]:
```
array([[   0,    0,    0, ...,  176, 3027,    3],
       [   0,    0,    0, ...,  163,  508, 1642],
       [   0,    0,    0, ...,   77,  240,   96],
       ...,
       [   0,    0,    0, ...,  328,    2,    4],
       [   0,    0,    0, ...,   65,   26,   16],
       [   4,   46,   25, ...,    2,    4,   16]])
```

In [14]:
```
# glove embeddings
lbl_target=LabelEncoder()
train_output=lbl_target.fit_transform(train_data['class'])
test_output=lbl_target.transform(test_data['class'])
```

In [15]:
```
#with open(r"C:\Users\91937\Downloads\glove.840B.300d.pkl.zip") as fp:
    #glove_embedding = pickle.load(fp)



import zipfile

with zipfile.ZipFile(r"C:\Users\91937\Downloads\glove.840B.300d.pkl.zip", 'r') a
    zip_ref.extractall(r"C:\Users\91937\Downloads")  # Specify your desired dire
```

In [16]:
```
import pickle

with open(r"C:\Users\91937\Downloads\glove.840B.300d.pkl", 'rb') as fp:
    glove_embedding = pickle.load(fp)
```

In [17]:
```
v=len(tokenizer.word_index)

embedding_matrix=np.zeros((v+1,300), dtype=float)
for word,idx in tokenizer.word_index.items():
    embedding_vector=glove_embedding.get(word)
    if embedding_vector is not None:
        embedding_matrix[idx]=embedding_vector
```

In [36]:
```
embedding_matrix
```

Out[36]:
```
array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.074482  ,  0.58293003, -0.78233999, ..., -0.24984001,
        -0.096953  ,  0.66692001],
       [-0.35394999,  0.23051   , -0.62689   , ..., -0.20720001,
         0.52003002,  0.51129001],
       ...,
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.29547   , -0.21822999, -0.039817  , ...,  0.62642998,
         0.48798001, -0.47554001],
       [ 0.75085002, -0.35099   ,  0.37674999, ..., -0.066863  ,
         0.79632998, -0.05967   ]])
```

In [38]:
```
early_stop=EarlyStopping(patience=5)
reducelr=ReduceLROnPlateau(patience=3)
```

Keras Sequential Model Construction

In [41]:
```
model=Sequential()
model.add(Input(shape=(40,)))
model.add(Embedding(v+1,300,weights=[embedding_matrix],trainable=False))
model.add(LSTM(20,return_sequences=True))
model.add(GlobalMaxPooling1D())
model.add(Dense(256,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(optimizer=keras.optimizers.SGD(0.1,momentum=0.09),loss='binary_crc
```

In [43]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape
embedding (Embedding)	(None, 40, 300)
lstm (LSTM)	(None, 40, 20)
global_max_pooling1d (GlobalMaxPooling1D)	(None, 20)
dense (Dense)	(None, 256)
dense_1 (Dense)	(None, 1)

```
Total params: 81,592,013 (311.25 MB)
Trainable params: 31,313 (122.32 KB)
Non-trainable params: 81,560,700 (311.13 MB)
```

Model Training and Evaluation

```
In [46]: r=model.fit(train_text_pad,train_output,validation_data=(test_text_pad,test_outp
                       epochs=20,batch_size=256,callbacks=[early_stop,reducelr])
```

```
Epoch 1/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 57s 68ms/step - accuracy: 0.7859 - loss: 0.4520 - va
l_accuracy: 0.8703 - val_loss: 0.3077 - learning_rate: 0.1000
Epoch 2/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 47s 65ms/step - accuracy: 0.8990 - loss: 0.2540 - va
l_accuracy: 0.9105 - val_loss: 0.2274 - learning_rate: 0.1000
Epoch 3/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 82s 65ms/step - accuracy: 0.9094 - loss: 0.2284 - va
l_accuracy: 0.9158 - val_loss: 0.2137 - learning_rate: 0.1000
Epoch 4/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 46s 63ms/step - accuracy: 0.9165 - loss: 0.2131 - va
l_accuracy: 0.9133 - val_loss: 0.2217 - learning_rate: 0.1000
Epoch 5/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 83s 64ms/step - accuracy: 0.9219 - loss: 0.2008 - va
l_accuracy: 0.9212 - val_loss: 0.2016 - learning_rate: 0.1000
Epoch 6/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 87s 70ms/step - accuracy: 0.9276 - loss: 0.1874 - va
l_accuracy: 0.9237 - val_loss: 0.1950 - learning_rate: 0.1000
Epoch 7/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 47s 64ms/step - accuracy: 0.9307 - loss: 0.1783 - va
l_accuracy: 0.9268 - val_loss: 0.1876 - learning_rate: 0.1000
Epoch 8/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 46s 64ms/step - accuracy: 0.9342 - loss: 0.1717 - va
l_accuracy: 0.9285 - val_loss: 0.1843 - learning_rate: 0.1000
Epoch 9/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 47s 64ms/step - accuracy: 0.9367 - loss: 0.1665 - va
l_accuracy: 0.9286 - val_loss: 0.1837 - learning_rate: 0.1000
Epoch 10/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 82s 64ms/step - accuracy: 0.9394 - loss: 0.1584 - va
l_accuracy: 0.9274 - val_loss: 0.1887 - learning_rate: 0.1000
Epoch 11/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 48s 66ms/step - accuracy: 0.9404 - loss: 0.1560 - va
l_accuracy: 0.9280 - val_loss: 0.1852 - learning_rate: 0.1000
Epoch 12/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 47s 64ms/step - accuracy: 0.9403 - loss: 0.1556 - va
l_accuracy: 0.9306 - val_loss: 0.1827 - learning_rate: 0.1000
Epoch 13/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 47s 64ms/step - accuracy: 0.9440 - loss: 0.1486 - va
l_accuracy: 0.9222 - val_loss: 0.2036 - learning_rate: 0.1000
Epoch 14/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 48s 66ms/step - accuracy: 0.9435 - loss: 0.1485 - va
l_accuracy: 0.9155 - val_loss: 0.2209 - learning_rate: 0.1000
Epoch 15/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 47s 64ms/step - accuracy: 0.9447 - loss: 0.1469 - va
l_accuracy: 0.9299 - val_loss: 0.1827 - learning_rate: 0.1000
Epoch 16/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 46s 63ms/step - accuracy: 0.9472 - loss: 0.1396 - va
l_accuracy: 0.9329 - val_loss: 0.1777 - learning_rate: 0.0100
Epoch 17/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 49s 67ms/step - accuracy: 0.9476 - loss: 0.1384 - va
l_accuracy: 0.9317 - val_loss: 0.1801 - learning_rate: 0.0100
Epoch 18/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 47s 65ms/step - accuracy: 0.9482 - loss: 0.1368 - va
l_accuracy: 0.9322 - val_loss: 0.1796 - learning_rate: 0.0100
Epoch 19/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 47s 65ms/step - accuracy: 0.9471 - loss: 0.1405 - va
l_accuracy: 0.9308 - val_loss: 0.1833 - learning_rate: 0.0100
Epoch 20/20
726/726 ━━━━━━━━━━━━━━━━━━━━ 47s 65ms/step - accuracy: 0.9470 - loss: 0.1389 - va
l_accuracy: 0.9317 - val_loss: 0.1805 - learning_rate: 1.0000e-03
```

```
In [ ]:
```

```
In [54]: from sklearn.metrics import classification_report

         # Adjust this line based on binary or multi-class approach
         predicted_classes = np.argmax(model.predict(test_text_pad), axis=1)  # or use bi

         print('TESTING DATA CLASSIFICATION REPORT \n \n')
         print(classification_report(test_output, predicted_classes,
                                     target_names=lbl_target.inverse_transform([0,1])))

         # For training data
         predicted_train_classes = np.argmax(model.predict(train_text_pad), axis=1)  # or

         print('TRAINING DATA CLASSIFICATION REPORT \n \n')
         print(classification_report(train_output, predicted_train_classes,
                                     target_names=lbl_target.inverse_transform([0,1])))
```

```
1451/1451 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step
TESTING DATA CLASSIFICATION REPORT 
 

              precision    recall  f1-score   support

 non-suicide       0.50      1.00      0.67     23209
     suicide       0.00      0.00      0.00     23206

    accuracy                           0.50     46415
   macro avg       0.25      0.50      0.33     46415
weighted avg       0.25      0.50      0.33     46415
```

```
C:\Users\91937\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.

C:\Users\91937\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.

C:\Users\91937\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
```

```
5802/5802 ━━━━━━━━━━━━━━━━━━━━ 69s 12ms/step
TRAINING DATA CLASSIFICATION REPORT 
 
```

```
C:\Users\91937\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
```

```
              precision    recall  f1-score   support

 non-suicide       0.50      1.00      0.67     92828
     suicide       0.00      0.00      0.00     92831

    accuracy                           0.50    185659
   macro avg       0.25      0.50      0.33    185659
weighted avg       0.25      0.50      0.33    185659
```

```
C:\Users\91937\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.

C:\Users\91937\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
```

In [64]:

```
twt = ['i am happy']
twt = tokenizer.texts_to_sequences(twt)
twt = pad_sequences(twt, maxlen=50)

prediction = model.predict(twt)[0][0]
print(prediction)

if(prediction > 0.5):
    print("Potential Suicide Post")
else:
    print("Non Suicide Post")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
0.3857283
Non Suicide Post
```

In [66]:

```
pickle.dump(tokenizer, open('tokenizer.pkl', 'wb'))
```

In [68]:

```
model.save("model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or `ke
ras.saving.save_model(model, 'my_model.keras')`.
```

In [70]:

```
token_form = pickle.load(open('tokenizer.pkl', 'rb'))
```

In [72]:

```
from keras.models import load_model
```

In [74]:

```
model_form = load_model("model.h5")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be b
uilt. `model.compile_metrics` will be empty until you train or evaluate the mode
l.
```

In [76]:

```
twt = ['Through these past years thoughts of suicide, fear, anxiety I’m so close
twt = token_form.texts_to_sequences(twt)
twt = pad_sequences(twt, maxlen=50)


prediction = model_form.predict(twt)[0][0]
print(prediction)

if(prediction > 0.5):
    print("Potential Suicide Post")
elif (prediction == 1):
    print("Non Suicide Post")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 916ms/step
0.9625426
Potential Suicide Post
```

In [ ]: