```
import pandas as pd
```

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

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

```
# Read the file into a DataFrame
file_path = '/content/drive/MyDrive/trainlstm.csv'  # Update the file path with your file location
train_df = pd.read_csv(file_path)

# Display the first few rows of the DataFrame
train_df.head()
```

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...
4	5	0	factsguide: society now #motivation

```
train_df['label'].value_counts()
```

```
0    29720
1     2242
Name: label, dtype: int64
```

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

```
--2024-03-23 01:42:19--  http://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|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.23MB/s    in 2m 39s

2024-03-23 01:44:59 (5.16 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]
```

```
!unzip glove.6B.zip
```

```
Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

```
import numpy as np

words = dict()

def add_to_dict(d, filename):
  with open(filename, 'r') as f:
    for line in f.readlines():
      line = line.split(' ')

      try:
        d[line[0]] = np.array(line[1:], dtype=float)
      except:
        continue

add_to_dict(words, 'glove.6B.50d.txt')
words
```

```
         0.57067  , -0.1036   ,  0.20422  ,  0.078308 , -0.42795  ,
        -1.7984   , -0.27865  ,  0.11954  , -0.12689  ,  0.031744 ,
         3.8631   , -0.17786  , -0.082434 , -0.62698  ,  0.26497  ,
        -0.057185 , -0.073521 ,  0.46103  ,  0.30862  ,  0.12498  ,
        -0.48609  , -0.0080272,  0.031184 , -0.36576  , -0.42699  ,
         0.42164  , -0.11666  , -0.50703  , -0.027273 , -0.53285  ]),
 'a': array([ 0.21705 ,  0.46515 , -0.46757 ,  0.10082 ,  1.0135  ,  0.74845 ,
        -0.53104 , -0.26256 ,  0.16812 ,  0.13182 , -0.24909 , -0.44185 ,
        -0.21739 ,  0.51004 ,  0.13448 , -0.43141 , -0.03123 ,  0.20674 ,
        -0.78138 , -0.20148 , -0.097401,  0.16088 , -0.61836 , -0.18504 ,
        -0.12461 , -2.2526  , -0.22321 ,  0.5043  ,  0.32257 ,  0.15313 ,
         3.9636  , -0.71365 , -0.67012 ,  0.28388 ,  0.21738 ,  0.14433 ,
         0.25926 ,  0.23434 ,  0.4274  , -0.44451 ,  0.13813 ,  0.36973 ,
        -0.64289 ,  0.024142, -0.039315, -0.26037 ,  0.12017 , -0.043782,
         0.41013 ,  0.1796  ]),
 '"': array([ 0.25769 ,  0.45629 , -0.76974 , -0.37679 ,  0.59272 , -0.063527,
         0.20545 , -0.57385 , -0.29009 , -0.13662 ,  0.32728 ,  1.4719  ,
        -0.73681 , -0.12036 ,  0.71354 , -0.46098 ,  0.65248 ,  0.48887 ,
        -0.51558 ,  0.039951, -0.34307 , -0.014087,  0.86488 ,  0.3546  ,
         0.7999  , -1.4995  , -1.8153  ,  0.41128 ,  0.23921 , -0.43139 ,
         3.6623  , -0.79834 , -0.54538 ,  0.16943 , -0.82017 , -0.3461  ,
         0.69495 , -1.2256  , -0.17992 , -0.057474,  0.030498, -0.39543 ,
        -0.38515 , -1.0002  ,  0.087599, -0.31009 , -0.34677 , -0.31438 ,
         0.75004 ,  0.97065 ]),
 "'s": array([ 0.23727  ,  0.40478  , -0.20547  ,  0.58805  ,  0.65533  ,
         0.32867  , -0.81964  , -0.23236  ,  0.27428  ,  0.24265  ,
         0.054992 ,  0.16296  , -1.2555   , -0.086437 ,  0.44536  ,
         0.096561 , -0.16519  ,  0.058378 , -0.38598  ,  0.086977 ,
         0.0033869,  0.55095  , -0.77697  , -0.62096  ,  0.092948 ,
        -2.5685   , -0.67739  ,  0.10151  , -0.48643  , -0.057805 ,
         3.1859   , -0.017554 , -0.16138  ,  0.055486 , -0.25885  ,
        -0.33938  , -0.19928  ,  0.26049  ,  0.10478  , -0.55934  ,
        -0.12342  ,  0.65961  , -0.51802  , -0.82995  , -0.082739 ,
         0.28155  , -0.423    , -0.27378  , -0.007901 , -0.030231 ]),
 'for': array([ 0.15272 ,  0.36181 , -0.22168 ,  0.066051,  0.13029 ,  0.37075 ,
        -0.75874 , -0.44722 ,  0.22563 ,  0.10208 ,  0.054225,  0.13494 ,
        -0.43052 , -0.2134  ,  0.56139 , -0.21445 ,  0.077974,  0.10137 ,
        -0.51306 , -0.40295 ,  0.40639 ,  0.23309 ,  0.20696 , -0.12668 ,
        -0.50634 , -1.7131  ,  0.077183, -0.39138 , -0.10594 , -0.23743 ,
         3.9552  ,  0.66596 , -0.61841 , -0.3268  ,  0.37021 ,  0.25764 ,
         0.38977 ,  0.27121 ,  0.043024, -0.34322 ,  0.020339,  0.2142  ,
         0.044097,  0.14003 , -0.20079 ,  0.074794, -0.36076 ,  0.43382 ,
        -0.084617,  0.1214  ]),
 '-': array([-0.16768  ,  1.2151   ,  0.49515  ,  0.26836  , -0.4585   ,
        -0.23311  , -0.52822  , -1.3557   ,  0.16098  ,  0.37691  ,
        -0.92702  , -0.43904  , -1.0634   ,  1.028    ,  0.0053943,
         0.04153  , -0.018638 , -0.55451  ,  0.026166 ,  0.28066  ,
        -0.66245  ,  0.23435  ,  0.2451   ,  0.025668 , -1.0869   ,
        -2.844    , -0.51272  ,  0.27286  ,  0.0071502,  0.033984 ,
         3.9084   ,  0.52766  , -0.66899  ,  1.8238   ,  0.43436  ,
        -0.30084  , -0.26996  ,  0.4394   ,  0.69956  ,  0.14885  ,
         0.029453 ,  1.4888   ,  0.52361  ,  0.099354 ,  1.2515   ,
         0.099381 , -0.079261 , -0.30862  ,  0.30893  ,  0.11023  ]),
 'that': array([ 0.88387  , -0.14199  ,  0.13566  ,  0.098682 ,  0.51218  ,
         0.49138  , -0.47155  , -0.30742  ,  0.01963  ,  0.12686  ,
```

```
len(words)
```

```
400000
```

```
import nltk

nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
tokenizer = nltk.RegexpTokenizer(r"\w+")

tokenizer.tokenize('@user when a father is dysfunctional and is')
```

```
['user', 'when', 'a', 'father', 'is', 'dysfunctional', 'and', 'is']
```

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

lemmatizer.lemmatize('feet')

def message_to_token_list(s):
  tokens = tokenizer.tokenize(s)
  lowercased_tokens = [t.lower() for t in tokens]
  lemmatized_tokens = [lemmatizer.lemmatize(t) for t in lowercased_tokens]
  useful_tokens = [t for t in lemmatized_tokens if t in words]

  return useful_tokens

message_to_token_list('@user feet a fathers is dysfunctional and is')
```

```
['user', 'foot', 'a', 'father', 'is', 'dysfunctional', 'and', 'is']
```

```
def message_to_word_vectors(message, word_dict=words):
  processed_list_of_tokens = message_to_token_list(message)

  vectors = []

  for token in processed_list_of_tokens:
    if token not in word_dict:
      continue

    token_vector = word_dict[token]
    vectors.append(token_vector)

  return np.array(vectors, dtype=float)
```

```
message_to_word_vectors('@user when a father is dysfunctional and is').shape
```

```
(8, 50)
```

```
train_df = train_df.sample(frac=1, random_state=1)
train_df.reset_index(drop=True, inplace=True)

split_index_1 = int(len(train_df) * 0.7)
split_index_2 = int(len(train_df) * 0.85)

train_df, val_df, test_df = train_df[:split_index_1], train_df[split_index_1:split_index_2], train_df[split_index_2:]

len(train_df), len(val_df), len(test_df)
```

```
(22373, 4794, 4795)
```

```
test_df
```

	id	label	tweet
27167	21271	0	thats how we do it. #homebrewpeeps
27168	26923	0	i havent ate no fast food/ home cooked food in...
27169	8332	0	i finally found a way how to delete old tweets...
27170	10079	0	because i'm happy clap along if you feel like ...
27171	24049	0	bye â□□ repost from @user be ! #kindness #hap...
...	...	...	...
31957	17290	0	remember itð□□□ #lost #empire #dreams #succes...
31958	5193	0	justice has been served #bosmatrial
31959	12173	0	ive just repurposed this former mustard jar in...
31960	236	0	the happiest baby ive ever knownð□□□ #cute #sm...
31961	29734	0	#ased bull up: you will dominate your bull a...

4795 rows × 3 columns

```
def df_to_X_y(dff):
  y = dff['label'].to_numpy().astype(int)

  all_word_vector_sequences = []

  for message in dff['tweet']:
    message_as_vector_seq = message_to_word_vectors(message)

    if message_as_vector_seq.shape[0] == 0:
      message_as_vector_seq = np.zeros(shape=(1, 50))

    all_word_vector_sequences.append(message_as_vector_seq)

  return all_word_vector_sequences, y
```

```
X_train, y_train = df_to_X_y(train_df)

print(len(X_train), len(X_train[0]))
```

```
22373 13
```

```
print(len(X_train), len(X_train[2]))
```

```
22373 7
```

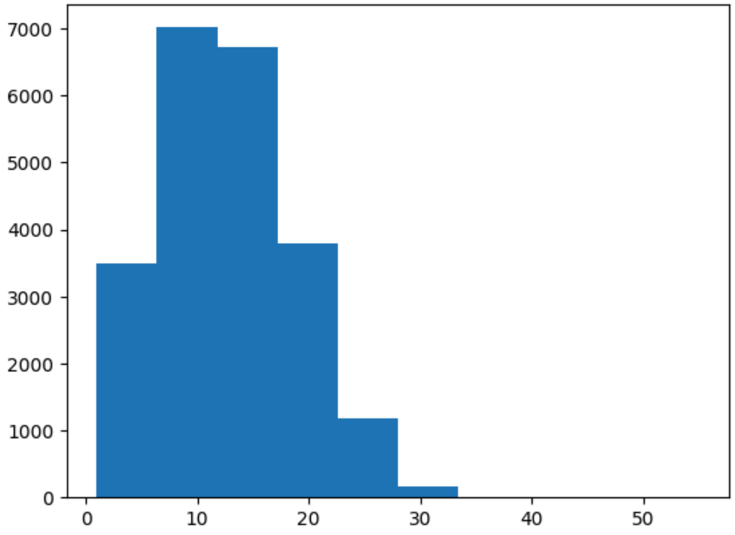
```
sequence_lengths = []

for i in range(len(X_train)):
  sequence_lengths.append(len(X_train[i]))

import matplotlib.pyplot as plt

plt.hist(sequence_lengths)
```

```
(array([3.493e+03, 7.017e+03, 6.723e+03, 3.786e+03, 1.182e+03, 1.610e+02,
        7.000e+00, 0.000e+00, 1.000e+00, 3.000e+00]),
 array([ 1. ,  6.4, 11.8, 17.2, 22.6, 28. , 33.4, 38.8, 44.2, 49.6, 55. ]),
 <BarContainer object of 10 artists>)
```



```
pd.Series(sequence_lengths).describe()
```

```
count    22373.000000
mean        12.692308
std          5.929912
min          1.000000
25%          8.000000
50%         12.000000
75%         17.000000
max         55.000000
dtype: float64
```

```
from copy import deepcopy

def pad_X(X, desired_sequence_length=57):
  X_copy = deepcopy(X)

  for i, x in enumerate(X):
    x_seq_len = x.shape[0]
    sequence_length_difference = desired_sequence_length - x_seq_len

    pad = np.zeros(shape=(sequence_length_difference, 50))

    X_copy[i] = np.concatenate([x, pad])

  return np.array(X_copy).astype(float)
```

```
X_train = pad_X(X_train)

X_train.shape
```

```
(22373, 57, 50)
```

```
y_train.shape
```

```
(22373,)
```

```
X_val, y_val = df_to_X_y(val_df)
X_val = pad_X(X_val)

X_val.shape, y_val.shape
```

```
((4794, 57, 50), (4794,))
```

```
X_test, y_test = df_to_X_y(test_df)
X_test = pad_X(X_test)

X_test.shape, y_test.shape
```

```
((4795, 57, 50), (4795,))
```

```
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

model = Sequential([])

model.add(layers.Input(shape=(57, 50)))
model.add(layers.LSTM(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.LSTM(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.LSTM(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 57, 64)            29440

 dropout (Dropout)           (None, 57, 64)            0

 lstm_1 (LSTM)               (None, 57, 64)            33024

 dropout_1 (Dropout)         (None, 57, 64)            0

 lstm_2 (LSTM)               (None, 57, 64)            33024

 dropout_2 (Dropout)         (None, 57, 64)            0

 flatten (Flatten)           (None, 3648)              0

 dense (Dense)               (None, 1)                 3649
```

```
=================================================================
Total params: 99137 (387.25 KB)
Trainable params: 99137 (387.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import AUC
from tensorflow.keras.callbacks import ModelCheckpoint

cp = ModelCheckpoint('model/', save_best_only=True)

model.compile(optimizer=Adam(learning_rate=0.0001),
              loss=BinaryCrossentropy(),
              metrics=['accuracy', AUC(name='auc')])
```

```
frequencies = pd.value_counts(train_df['label'])

frequencies
```

```
0    20820
1     1553
Name: label, dtype: int64
```

```
weights = {0: frequencies.sum() / frequencies[0], 1: frequencies.sum() / frequencies[1]}
weights
```

```
{0: 1.0745917387127761, 1: 14.406310367031551}
```

```
model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=20, callbacks=[cp], class_weight=weights)
```

```
Epoch 1/20
700/700 [==============================] - 34s 34ms/step - loss: 0.9951 - accuracy: 0.7405 - auc: 0.8390 - val_loss: 0.3809 - val_accura
Epoch 2/20
700/700 [==============================] - 14s 20ms/step - loss: 0.8243 - accuracy: 0.8142 - auc: 0.8935 - val_loss: 0.4562 - val_accura
Epoch 3/20
700/700 [==============================] - 9s 13ms/step - loss: 0.7721 - accuracy: 0.8284 - auc: 0.9072 - val_loss: 0.4308 - val_accurac
Epoch 4/20
700/700 [==============================] - 13s 18ms/step - loss: 0.7556 - accuracy: 0.8327 - auc: 0.9112 - val_loss: 0.4260 - val_accura
Epoch 5/20
700/700 [==============================] - 20s 29ms/step - loss: 0.7337 - accuracy: 0.8335 - auc: 0.9166 - val_loss: 0.3699 - val_accura
Epoch 6/20
700/700 [==============================] - 12s 17ms/step - loss: 0.7159 - accuracy: 0.8411 - auc: 0.9207 - val_loss: 0.4470 - val_accura
Epoch 7/20
700/700 [==============================] - 9s 13ms/step - loss: 0.6971 - accuracy: 0.8414 - auc: 0.9248 - val_loss: 0.4461 - val_accurac
Epoch 8/20
700/700 [==============================] - 22s 31ms/step - loss: 0.6765 - accuracy: 0.8475 - auc: 0.9294 - val_loss: 0.3285 - val_accura
Epoch 9/20
700/700 [==============================] - 22s 31ms/step - loss: 0.6568 - accuracy: 0.8477 - auc: 0.9334 - val_loss: 0.2733 - val_accura
Epoch 10/20
700/700 [==============================] - 11s 15ms/step - loss: 0.6427 - accuracy: 0.8522 - auc: 0.9361 - val_loss: 0.3016 - val_accura
Epoch 11/20
700/700 [==============================] - 12s 18ms/step - loss: 0.6301 - accuracy: 0.8511 - auc: 0.9386 - val_loss: 0.3838 - val_accura
Epoch 12/20
700/700 [==============================] - 9s 13ms/step - loss: 0.6145 - accuracy: 0.8533 - auc: 0.9413 - val_loss: 0.3653 - val_accurac
Epoch 13/20
700/700 [==============================] - 13s 19ms/step - loss: 0.6034 - accuracy: 0.8588 - auc: 0.9437 - val_loss: 0.3758 - val_accura
Epoch 14/20
700/700 [==============================] - 10s 14ms/step - loss: 0.5843 - accuracy: 0.8582 - auc: 0.9468 - val_loss: 0.3333 - val_accura
Epoch 15/20
700/700 [==============================] - 12s 18ms/step - loss: 0.5671 - accuracy: 0.8637 - auc: 0.9499 - val_loss: 0.3322 - val_accura
Epoch 16/20
700/700 [==============================] - 11s 16ms/step - loss: 0.5737 - accuracy: 0.8602 - auc: 0.9485 - val_loss: 0.3282 - val_accura
Epoch 17/20
700/700 [==============================] - 23s 32ms/step - loss: 0.5479 - accuracy: 0.8632 - auc: 0.9527 - val_loss: 0.2679 - val_accura
Epoch 18/20
700/700 [==============================] - 10s 14ms/step - loss: 0.5369 - accuracy: 0.8676 - auc: 0.9546 - val_loss: 0.2940 - val_accura
Epoch 19/20
700/700 [==============================] - 13s 19ms/step - loss: 0.5305 - accuracy: 0.8683 - auc: 0.9556 - val_loss: 0.3888 - val_accura
Epoch 20/20
700/700 [==============================] - 9s 13ms/step - loss: 0.5081 - accuracy: 0.8715 - auc: 0.9590 - val_loss: 0.4817 - val_accurac
<keras.src.callbacks.History at 0x7c9d4836a650>
```

```
from tensorflow.keras.models import load_model

best_model = load_model('model/')
```

```
test_predictions = (best_model.predict(X_test) > 0.5).astype(int)

from sklearn.metrics import classification_report

print(classification_report(y_test, test_predictions))
```

```
150/150 [==============================] - 2s 5ms/step
              precision    recall  f1-score   support

           0       0.99      0.88      0.93      4454
           1       0.35      0.85      0.50       341

    accuracy                           0.88      4795
   macro avg       0.67      0.87      0.72      4795
weighted avg       0.94      0.88      0.90      4795
```