Bi-LSTM And BERT Model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
```

The dataset used for the project is then loaded from the csv file into a pandas dataframe.

```
counseldf = pd.read_csv('counselchat-data.csv')
```

The below function prints the columns available within the dataset.

```
print(counseldf.columns)
```

```
Index(['questionID', 'questionTitle', 'questionText', 'questionUrl', 'topics',
       'therapistName', 'therapistUrl', 'answerText', 'upvotes'],
      dtype='object')
```

Below the first 5 samples within the dataset are displayed.

```
counseldf.head()
```

	questionID	questionTitle	questionText	questionUrl	topics	therapistName	
0	5566fab2a64752d71ec3ca69	Escalating disagreements between mother and wife	My wife and mother are having tense disagreeme...	https://counselchat.com/questions/escalating-d...	Family Conflict	Kristi King-Morgan, LMSW	htt
1	5566f94fa64752d71ec3ca64	I'm addicted to smoking. How can I stop?	I'm planning to have baby, so I have to quit s...	https://counselchat.com/questions/i-m-addicted...	Substance Abuse,Addiction	Rebecca Duellman	https:/
2	5567d26887a1cc0c3f3d8f46	Keeping secrets from my family	I have secrets in my mind, and I don't know wh...	https://counselchat.com/questions/keeping-secr...	Family Conflict	Jeevna Bajaj	https
3	556bed15c969ba5861709df5	The Underlying Causes of Being Possessive	I am extremely possessive in my relationships ...	https://counselchat.com/questions/the-underlyi...	Behavioral Change,Social Relationships	Rebecca Duellman	https:/
4	556ba115c969ba5861709de6	Can I control anxiety without medication?	I had a head injury a few years ago and my min...	https://counselchat.com/questions/can-i-contro...	Anxiety	Rebecca Duellman	https:/

As the dataset contains some columns with personal information about the therapist, those columns are removed. Moreover, for the purposes of this project the only relevant columns are 'questionText' and 'topics'. Hence all other columns except those are removed before further processing.

```
counseldf.drop(['questionID', 'questionTitle', 'questionUrl', 'therapistName', 'therapistUrl','answerText', 'upvotes'], axis = 1, inplac
```

```
counseldf.head()
```

	questionText	topics
0	My wife and mother are having tense disagreeme...	Family Conflict
1	I'm planning to have baby, so I have to quit s...	Substance Abuse,Addiction
2	I have secrets in my mind, and I don't know wh...	Family Conflict
3	I am extremely possessive in my relationships ...	Behavioral Change,Social Relationships
4	I had a head injury a few years ago and my min...	Anxiety

```
counseldf.shape
```

```
(1482, 2)
```

```
counseldf.isnull().sum()
```

Add text cell

```
questionTe
topics          10
dtype: int64
```

The datset contains some null values in both columns. To avoid the problem created by null values, the samples containing them are simply removed.

```
counseldf = counseldf.dropna(axis=0)
```

```
counseldf.isnull().sum()
```

```
questionText    0
topics          0
dtype: int64
```

```
counseldf.shape
```

```
(1376, 2)
```

Finally, we have the clean dataset which can now be processed with NLP techniques. After removing rows with null values, the size of the dataset remains at 1376 samples.

The 'topics' column is the target categories for the samples. As seen in the first few lines, some of the samples contain multiple labels. An LSTM network was tried to train using multiple labels but as the model failed to converge, it was decided that only the first label would be kept as the target label of the sample if it has multiple labels. The below code does just that.

```
counseldf['topics'] = counseldf['topics'].str.split(',')
counseldf['topics'] = counseldf['topics'].apply(lambda x: x[0])
```

```
counseldf.head()
```

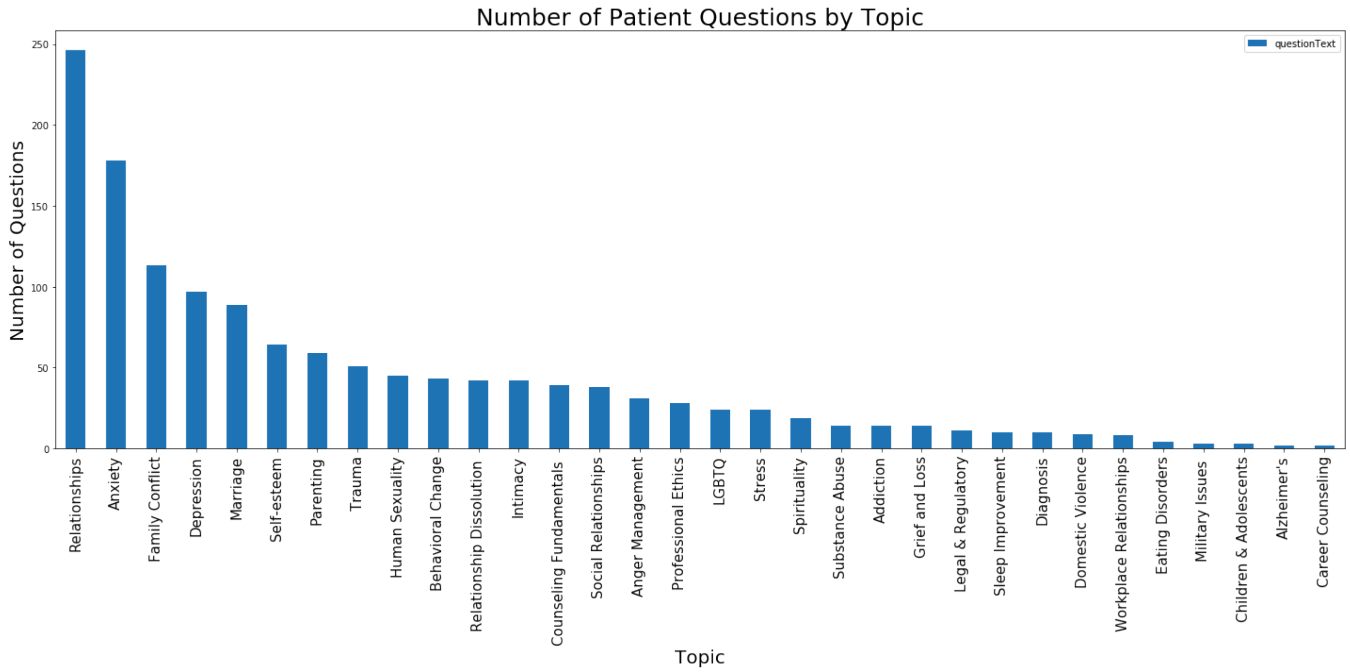
	questionText	topics
0	My wife and mother are having tense disagreeme...	Family Conflict
1	I'm planning to have baby, so I have to quit s...	Substance Abuse
2	I have secrets in my mind, and I don't know wh...	Family Conflict
3	I am extremely possessive in my relationships ...	Behavioral Change
4	I had a head injury a few years ago and my min...	Anxiety

```
counseldf['topics'].value_counts()
```

```
Relationships                246
Anxiety                      178
Family Conflict              113
Depression                    97
Marriage                      89
Self-esteem                   64
Parenting                     59
Trauma                        51
Human Sexuality               45
Behavioral Change             43
Relationship Dissolution      42
Intimacy                      42
Counseling Fundamentals       39
Social Relationships          38
Anger Management              31
Professional Ethics           28
LGBTQ                         24
Stress                        24
Spirituality                  19
Grief and Loss                14
Substance Abuse               14
Addiction                     14
Legal & Regulatory            11
Sleep Improvement             10
Diagnosis                     10
Domestic Violence              9
Workplace Relationships        8
Eating Disorders               4
```

```
Military Issues              3
Children & Adolescents       3
Career Counseling            2
Alzheimer's                  2
Name: topics, dtype: int64
```

```
fig, ax = plt.s Add text cell ize=(20, 10))
counseldf.groupby('topics').agg('count').sort_values('questionText', ascending=False).plot.bar(ax=ax)
ax.set_title("Number of Patient Questions by Topic", fontsize=25)
ax.set_ylabel("Number of Questions", fontsize=20)
ax.set_xlabel("Topic", fontsize=20)
ax.set_xticklabels(ax.get_xticklabels(), fontsize=15)
plt.tight_layout()
plt.show()
```



After this initial analysis to look at the overall dataset, the 'topics' column could be one-hot encoded so that it is ready for the learning model to take as targets.

```
targetdf = pd.get_dummies(counseldf['topics'])
targetdf.head()
```

	Addiction	Alzheimer's	Anger Management	Anxiety	Behavioral Change	Career Counseling	Children & Adolescents	Counseling Fundamentals	Depression	Diagnosis	...	Relati Disc
0	0	0	0	0	0	0	0	0	0	0	...	
1	0	0	0	0	0	0	0	0	0	0	...	
2	0	0	0	0	0	0	0	0	0	0	...	
3	0	0	0	0	1	0	0	0	0	0	...	
4	0	0	0	1	0	0	0	0	0	0	...	

5 rows × 32 columns

Now, finally it is time to apply natural language processing techniques to each of the individual text samples.

First, we define a pre-process function.

```
#Preprocess function


import nltk, re
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
from nltk.corpus import wordnet
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from collections import Counter

stop_words = stopwords.words('english')
normalizer = WordNetLemmatizer()

def get_part_of_speech(word):
    probable_part_of_speech = wordnet.synsets(word)
    pos_counts = Counter()
    pos_counts["n"] = len(  [ item for item in probable_part_of_speech if item.pos()=="n"]  )
    pos_counts["v"] = len(  [ item for item in probable_part_of_speech if item.pos()=="v"]  )
    pos_counts["a"] = len(  [ item for item in probable_part_of_speech if item.pos()=="a"]  )
    pos_counts["r"] = len(  [ item for item in probable_part_of_speech if item.pos()=="r"]  )
    most_likely_part_of_speech = pos_counts.most_common(1)[0][0]
    return most_likely_part_of_speech

def preprocess_text(text):
    cleaned = re.sub(r'\W+', ' ', text).lower()
    tokenized = word_tokenize(cleaned)
    normalized = [normalizer.lemmatize(token, get_part_of_speech(token)) for token in tokenized]
    return normalized
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

The few cells below show the NLP pre-processing techniques applied to the first samples of text.

```
counseldf['questionText'][0]
```

```
'My wife and mother are having tense disagreements. In the past, they’ve had minor differences. For example, my wife would complain
to me my mother is too overbearing; my mother would complain my wife is lazy.\r\nHowever, it’s intensified lately. I think the
cause is my wife talked back to her once. Now, any little disagreement is magnified, leading to major disagreements. What can I
do?'
```

```
text = counseldf['questionText'][0]
cleaned = re.sub(r'\W+', ' ', text).lower()
print(cleaned)
```

```
my wife and mother are having tense disagreements in the past they ve had minor differences for example my wife would complain to me
```

```
tokenized = word_tokenize(cleaned)
print(tokenized)
```

```
['my', 'wife', 'and', 'mother', 'are', 'having', 'tense', 'disagreements', 'in', 'the', 'past', 'they', 've', 'had', 'minor', 'diffe
```

```
normalized = [normalizer.lemmatize(token, get_part_of_speech(token)) for token in tokenized]
print(normalized)
```

```
['my', 'wife', 'and', 'mother', 'be', 'have', 'tense', 'disagreement', 'in', 'the', 'past', 'they', 've', 'have', 'minor', 'differer
```

It is clear from the above few cells what pre-processing does to each of the samples. Below the same process is applied to the entire dataset calling the pre-process function defined above.

```
processed_questionText = counseldf['questionText'].apply(lambda x: preprocess_text(x))
```

After pre-processing, stop words removal can be done to each of the samples.

```
stop_words = set(stopwords.words('english'))

questionText_nostops = []
for title in processed_questionText:
    text_no_stops = [word for word in title if word not in stop_words]
    questionTe  Add text cell  pend(text_no_stops)
```

Below is the first sample with stop words removed.

```
print(questionText_nostops[0])
```

```
['wife', 'mother', 'tense', 'disagreement', 'past', 'minor', 'difference', 'example', 'wife', 'would', 'complain', 'mother', 'overbe
```

Finally, with the tokens available for all samples, it is time to create the word-index dictionary for changing the data to numerical values. To do this, Tensorflow's tokenizer class will be used.

```
import tensorflow as tf
from tensorflow.keras import preprocessing

tf.random.set_seed(4)
tf.__version__
```

```
'2.2.0'
```

```
tokenizer = preprocessing.text.Tokenizer()
tokenizer.fit_on_texts( questionText_nostops )
tokenized_questions = tokenizer.texts_to_sequences( questionText_nostops )
print('Sample tokenized: {}'.format(tokenized_questions[0]))
print('==================================================================================================================\n')

length_list = list()
for token_seq in tokenized_questions:
    length_list.append( len( token_seq ))
max_input_length = np.array( length_list ).max()
print( 'Questions max length is {} words'.format( max_input_length ))
print('==================================================================================================================\n')

padded_questions = preprocessing.sequence.pad_sequences( tokenized_questions , maxlen=max_input_length , padding='post' )
input_data = np.array( padded_questions )
print( 'Input data shape -> {}\n'.format( input_data.shape ))
print('Input data sample->\n {}'.format(input_data[0]))
print('==================================================================================================================\n')

question_word_dict = tokenizer.word_index
num_question_tokens = len( question_word_dict )+1
print( 'Number of Question tokens = {}\n'.format( num_question_tokens))
print('Dictionary: {}'.format(question_word_dict))
```

```
Sample tokenized: [68, 101, 1760, 971, 49, 858, 972, 684, 68, 52, 477, 101, 1761, 101, 52, 477, 68, 859, 165, 1386, 150, 9, 216, 68,
==================================================================================================================

Questions max length is 220 words
==================================================================================================================

Input data shape -> (1376, 220)

Input data sample->
 [  68  101 1760  971   49  858  972  684   68   52  477  101 1761  101
   52  477   68  859  165 1386  150    9  216   68   18   41  166  971
 1762  353  626  971    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0]
==================================================================================================================

Number of Question tokens = 2417

Dictionary: {'feel': 1, 'get': 2, 'want': 3, 'like': 4, 'know': 5, 'time': 6, 'go': 7, 'year': 8, 'think': 9, 'say': 10, 'make': 11,
```

Finally, with the input data ready with the numerical token indices, it is time to split the dataset into training and testing sets.

```
from sklearn.mo Add text cell on import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(input_data, targetdf, test_size = 0.3, random_state=2)
```

```
print(x_train.shape, x_test.shape)
print(y_train.shape, y_test.shape)
```

```
(963, 220) (413, 220)
(963, 32) (413, 32)
```

Bi-directional LSTM Model

```
inputs = tf.keras.layers.Input(shape=(None,))
embedding = tf.keras.layers.Embedding(num_question_tokens, 200, mask_zero=True)(inputs)
lstm_outputs = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(100, return_state=False), name='bidir') (embedding)
dense_outputs = tf.keras.layers.Dense(500, activation=tf.keras.activations.relu) (lstm_outputs)
outputs = tf.keras.layers.Dense(32, activation=tf.keras.activations.softmax) (dense_outputs)

LSTM_model = tf.keras.models.Model(inputs, outputs)
LSTM_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy', 'Precision', 'Recall'])
```

```
LSTM_model.summary()
```

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, None)]            0
_________________________________________________________________
embedding_2 (Embedding)      (None, None, 200)         483400
_________________________________________________________________
bidir (Bidirectional)        (None, 200)               240800
_________________________________________________________________
dense_5 (Dense)              (None, 500)               100500
_________________________________________________________________
dense_6 (Dense)              (None, 32)                16032
=================================================================
Total params: 840,732
Trainable params: 840,732
Non-trainable params: 0
_________________________________________________________________
```

The summary of the model is displayed above which shows a total of 840,732 parameters. In below cell, the compiled model is trained upto 50 epochs.

```
history = LSTM_model.fit(x_train, y_train, validation_data=(x_test, y_test), batch_size=128, epochs=50, verbose=1)
```

```
Epoch 1/50
8/8 [==============================] - 13s 2s/step - loss: 3.4293 - accuracy: 0.1931 - precision: 0.0000e+00 - recall: 0.0000e+0
Epoch 2/50
8/8 [==============================] - 13s 2s/step - loss: 3.0861 - accuracy: 0.1838 - precision: 0.3023 - recall: 0.0135 - val_
Epoch 3/50
8/8 [==============================] - 14s 2s/step - loss: 2.7802 - accuracy: 0.2534 - precision: 0.0000e+00 - recall: 0.0000e+0
Epoch 4/50
8/8 [==============================] - 14s 2s/step - loss: 2.5925 - accuracy: 0.2440 - precision: 0.9333 - recall: 0.0145 - val_
Epoch 5/50
8/8 [==============================] - 14s 2s/step - loss: 2.3091 - accuracy: 0.3198 - precision: 0.9792 - recall: 0.0976 - val_
Epoch 6/50
8/8 [==============================] - 15s 2s/step - loss: 1.9724 - accuracy: 0.4299 - precision: 0.8782 - recall: 0.2471 - val_
Epoch 7/50
8/8 [==============================] - 15s 2s/step - loss: 1.6289 - accuracy: 0.5234 - precision: 0.9368 - recall: 0.3697 - val_
Epoch 8/50
8/8 [==============================] - 15s 2s/step - loss: 1.2777 - accuracy: 0.6064 - precision: 0.9663 - recall: 0.4766 - val_
Epoch 9/50
8/8 [==============================] - 15s 2s/step - loss: 0.9632 - accuracy: 0.7404 - precision: 0.9810 - recall: 0.5909 - val_
Epoch 10/50
8/8 [==============================] - 16s 2s/step - loss: 0.6931 - accuracy: 0.8224 - precision: 0.9824 - recall: 0.6968 - val_
Epoch 11/50
8/8 [==============================] - 16s 2s/step - loss: 0.4848 - accuracy: 0.8775 - precision: 0.9884 - recall: 0.7934 - val_
Epoch 12/50
8/8 [==============================] - 16s 2s/step - loss: 0.4230 - accuracy: 0.8827 - precision: 0.9743 - recall: 0.8276 - val_
Epoch 13/50
8/8 [==============================] - 16s 2s/step - loss: 0.7296 - accuracy: 0.8255 - precision: 0.9106 - recall: 0.7508 - val_
Epoch 14/50
```

```
8/8 [==============================] - 16s 2s/step - loss: 0.4267 - accuracy: 0.9013 - precision: 0.9725 - recall: 0.8089 - val_
Epoch 15/50
8/8 [==============================] - 17s 2s/step - loss: 0.3053 - accuracy: 0.9367 - precision: 0.9905 - recall: 0.8702 - val_
Epoch 16/50
8/8 [==============================] - 17s 2s/step - loss: 0.2285 - accuracy: 0.9574 - precision: 1.0000 - recall: 0.9107 - val_
Epoch 17/50
8/8 [==============================] - 17s 2s/step - loss: 0.1729 - accuracy: 0.9678 - precision: 1.0000 - recall: 0.9315 - val_
Epoch 18/50
8/8 [==============================] - 17s 2s/step - loss: 0.1381 - accuracy: 0.9740 - precision: 1.0000 - recall: 0.9429 - val_
Epoch 19/50
8/8 [==============================] - 17s 2s/step - loss: 0.1110 - accuracy: 0.9834 - precision: 1.0000 - recall: 0.9543 - val_
Epoch 20/50
8/8 [==============================] - 17s 2s/step - loss: 0.0897 - accuracy: 0.9875 - precision: 1.0000 - recall: 0.9595 - val_
Epoch 21/50
8/8 [==============================] - 18s 2s/step - loss: 0.0734 - accuracy: 0.9886 - precision: 1.0000 - recall: 0.9657 - val_
Epoch 22/50
8/8 [==============================] - 18s 2s/step - loss: 0.0597 - accuracy: 0.9896 - precision: 1.0000 - recall: 0.9730 - val_
Epoch 23/50
8/8 [==============================] - 18s 2s/step - loss: 0.0493 - accuracy: 0.9896 - precision: 1.0000 - recall: 0.9813 - val_
Epoch 24/50
8/8 [==============================] - 18s 2s/step - loss: 0.0411 - accuracy: 0.9896 - precision: 1.0000 - recall: 0.9844 - val_
Epoch 25/50
8/8 [==============================] - 18s 2s/step - loss: 0.0347 - accuracy: 0.9927 - precision: 1.0000 - recall: 0.9875 - val_
Epoch 26/50
8/8 [==============================] - 17s 2s/step - loss: 0.0290 - accuracy: 0.9969 - precision: 1.0000 - recall: 0.9875 - val_
Epoch 27/50
8/8 [==============================] - 18s 2s/step - loss: 0.0250 - accuracy: 0.9979 - precision: 1.0000 - recall: 0.9927 - val_
Epoch 28/50
8/8 [==============================] - 19s 2s/step - loss: 0.0210 - accuracy: 0.9990 - precision: 1.0000 - recall: 0.9948 - val_
```

```
LSTM_model.save('LSTM_model.h5')
```

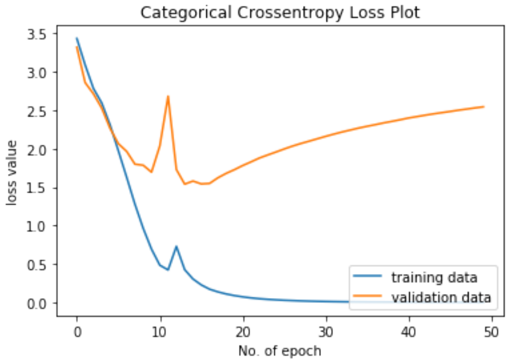
```
history_df = pd.DataFrame(history.history)
history_df['f1_score'] = (2 * history_df['precision']*history_df['recall'])/(history_df['precision']+history_df['recall'])
history_df['val_f1_score'] = (2 * history_df['val_precision']*history_df['val_recall'])/(history_df['val_precision']+history_df['val_re
```

```
history_df
```

	loss	accuracy	precision	recall	val_loss	val_accuracy	val_precision	val_recall	f1_score	val_f1_score
0	3.429279	0.193146	0.000000	0.000000	3.315263	0.203390	0.000000	0.000000	NaN	NaN
1	3.086149	0.183801	0.302326	0.013499	2.856819	0.181598	0.000000	0.000000	0.025845	NaN
2	2.780[illegible]	[illegible]5	0.000000	0.000000	2.711107	0.232446	0.000000	0.000000	NaN	NaN
3	2.592494	0.244029	0.933333	0.014538	2.525123	0.314770	1.000000	0.038741	0.028630	0.074592
4	2.309086	0.319834	0.979167	0.097612	2.263654	0.326877	0.777778	0.220339	0.177526	0.343396
5	1.972432	0.429907	0.878229	0.247144	2.063815	0.447942	0.871795	0.246973	0.385737	0.384906
6	1.628857	0.523364	0.936842	0.369678	1.963772	0.472155	0.887417	0.324455	0.530156	0.475177
7	1.277683	0.606438	0.966316	0.476636	1.797178	0.549637	0.867347	0.411622	0.638387	0.558292
8	0.963197	0.740395	0.981034	0.590862	1.783234	0.615012	0.870690	0.489104	0.737524	0.626357
9	0.693108	0.822430	0.982430	0.696781	1.693442	0.631961	0.873469	0.518160	0.815310	0.650456
10	0.484828	0.877466	0.988357	0.793354	2.033668	0.639225	0.788779	0.578692	0.880184	0.667598
11	0.423022	0.882658	0.974328	0.827622	2.680096	0.583535	0.702265	0.525424	0.895003	0.601108
12	0.729555	0.825545	0.910579	0.750779	1.727211	0.653753	0.797342	0.581114	0.822994	0.672269
13	0.426747	0.901350	0.972534	0.808930	1.537615	0.663438	0.836120	0.605327	0.883220	0.702247
14	0.305344	0.936656	0.990544	0.870197	1.577577	0.685230	0.809231	0.636804	0.926479	0.712737
15	0.228515	0.957425	1.000000	0.910696	1.540629	0.714286	0.842271	0.646489	0.953261	0.731507
16	0.172882	0.967809	1.000000	0.931464	1.546815	0.711864	0.842424	0.673123	0.964516	0.748318
17	0.138115	0.974039	1.000000	0.942887	1.619801	0.726392	0.829912	0.685230	0.970604	0.750663
18	0.110994	0.983385	1.000000	0.954309	1.677581	0.731235	0.830946	0.702179	0.976621	0.761155
19	0.089713	0.987539	1.000000	0.959502	1.725805	0.731235	0.809524	0.699758	0.979332	0.750649
20	0.073351	0.988577	1.000000	0.965732	1.779573	0.728814	0.812155	0.711864	0.982567	0.758710
21	0.059709	0.989616	1.000000	0.973001	1.826932	0.731235	0.804878	0.719128	0.986316	0.759591
22	0.049271	0.989616	1.000000	0.981308	1.876773	0.728814	0.803235	0.721550	0.990566	0.760204
23	0.041133	0.989616	1.000000	0.984424	1.916360	0.728814	0.796791	0.721550	0.992151	0.757306
24	0.034701	0.992731	1.000000	0.987539	1.954810	0.731235	0.797333	0.723971	0.993730	0.758883
25	0.029006	0.996885	1.000000	0.987539	1.994534	0.733656	0.790451	0.721550	0.993730	0.754430
26	0.024988	0.997923	1.000000	0.992731	2.031993	0.738499	0.797872	0.726392	0.996352	0.760456
27	0.020967	0.998962	1.000000	0.994808	2.065400	0.736077	0.793103	0.723971	0.997397	0.756962
28	0.017871	0.998962	1.000000	0.995846	2.095803	0.738499	0.793194	0.733656	0.997919	0.762264
29	0.015238	0.998962	1.000000	0.997923	2.127288	0.738499	0.788512	0.731235	0.998960	0.758794
30	0.013029	0.998962	1.000000	0.997923	2.158432	0.736077	0.786458	0.731235	0.998960	0.757842
31	0.011431	0.998962	1.000000	0.997923	2.187973	0.738499	0.782383	0.731235	0.998960	0.755945
32	0.009887	0.998962	1.000000	0.997923	2.215883	0.738499	0.784974	0.733656	0.998960	0.758448
33	0.008687	0.998962	1.000000	0.997923	2.241267	0.738499	0.778920	0.733656	0.998960	0.755611
34	0.007669	1.000000	1.000000	0.997923	2.266715	0.738499	0.769821	0.728814	0.998960	0.748756
35	0.006874	1.000000	1.000000	0.997923	2.289507	0.736077	0.775773	0.728814	0.998960	0.751561
36	0.006198	1.000000	1.000000	0.997923	2.310653	0.736077	0.775773	0.728814	0.998960	0.751561
37	0.005740	1.000000	1.000000	0.997923	2.333482	0.738499	0.771795	0.728814	0.998960	0.749689
38	0.005119	1.000000	1.000000	0.998962	2.353501	0.738499	0.771795	0.728814	0.999481	0.749689
39	0.004677	1.000000	1.000000	0.998962	2.374531	0.738499	0.771795	0.728814	0.999481	0.749689
40	0.004306	1.000000	1.000000	0.998962	2.397133	0.738499	0.770408	0.731235	0.999481	0.750311
41	0.003962	1.000000	1.000000	0.998962	2.415037	0.738499	0.768448	0.731235	0.999481	0.749380
42	0.003700	1.000000	1.000000	0.998962	2.433464	0.738499	0.770408	0.731235	0.999481	0.750311
43	0.003427	1.000000	1.000000	0.998962	2.450770	0.738499	0.770408	0.731235	0.999481	0.750311
44	0.003173	1.000000	1.000000	0.998962	2.466246	0.738499	0.770408	0.731235	0.999481	0.750311
45	0.002973	1.000000	1.000000	0.998962	2.481681	0.738499	0.769036	0.733656	0.999481	0.750929
46	0.002800	1.000000	1.000000	1.000000	2.498541	0.738499	0.769036	0.733656	1.000000	0.750929
47	0.002618	1.000000	1.000000	1.000000	2.512894	0.738499	0.769036	0.733656	1.000000	0.750929
48	0.002465	1.000000	1.000000	1.000000	2.528027	0.736077	0.769036	0.733656	1.000000	0.750929

49	0.002321	1.000000	1.000000	1.000000	2.541599	0.733656	0.768448	0.731235	1.000000	0.749380

```
plt.plot(history.history['loss'], label='training data')
plt.plot(history.history['val_loss'], label='validation data')
plt.title('Cate Add text cell ssentropy Loss Plot')
plt.ylabel('los
plt.xlabel('No. of epoch')
plt.legend(loc="lower right")
plt.show()
```



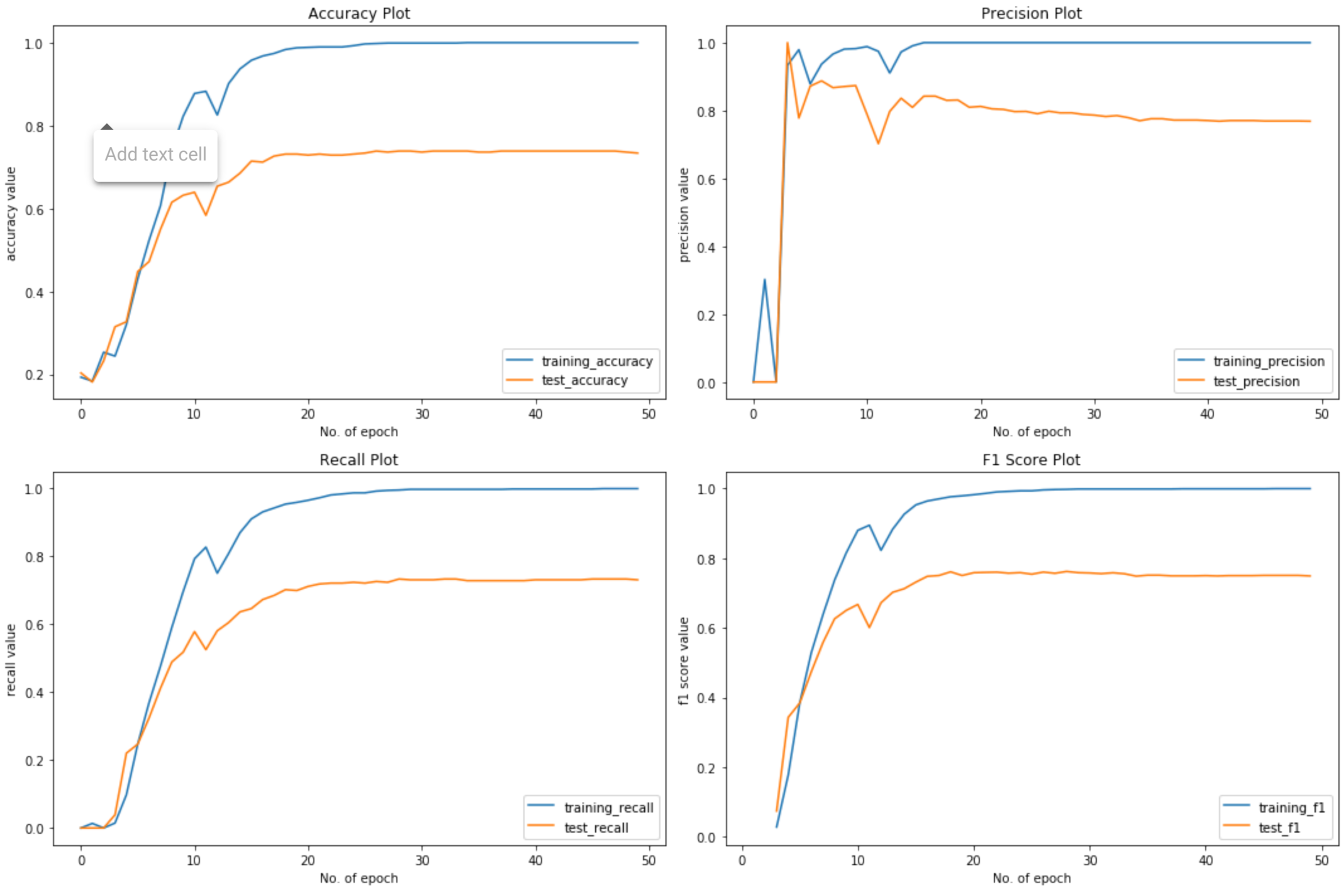

```
plt.subplots(figsize=(15,10))
ax = plt.subplot(2,2,1)
plt.plot(history.history['accuracy'], label='training_accuracy')
plt.plot(history.history['val_accuracy'], label='test_accuracy')
plt.title('Accuracy Plot')
plt.ylabel('accuracy value')
plt.xlabel('No. of epoch')
plt.legend(loc="lower right")

ax = plt.subplot(2,2,2)
plt.plot(history.history['precision'], label='training_precision')
plt.plot(history.history['val_precision'], label='test_precision')
plt.title('Precision Plot')
plt.ylabel('precision value')
plt.xlabel('No. of epoch')
plt.legend(loc="lower right")

ax = plt.subplot(2,2,3)
plt.plot(history.history['recall'], label='training_recall')
plt.plot(history.history['val_recall'], label='test_recall')
plt.title('Recall Plot')
plt.ylabel('recall value')
plt.xlabel('No. of epoch')
plt.legend(loc="lower right")

ax = plt.subplot(2,2,4)
plt.plot(history_df['f1_score'], label='training_f1')
plt.plot(history_df['val_f1_score'], label='test_f1')
plt.title('F1 Score Plot')
plt.ylabel('f1 score value')
plt.xlabel('No. of epoch')
plt.legend(loc="lower right")

plt.tight_layout()
plt.show()
```



Below, a predict function is defined that will take as input natural language text and try to classify it into the categories of mental health issues defined from the dataset.

```
def predict(text):
    processed = preprocess_text(text)
    text_no_stops = [word for word in processed if word not in stop_words]
    tokenized = tokenizer.texts_to_sequences( text_no_stops )
    padded = preprocessing.sequence.pad_sequences( tokenized , maxlen=max_input_length , padding='post' )
    input_data = np.array( padded )
    output = model.predict(input_data)
    index = np.argmax(output[0])
    print(targetdf.columns[index])
```

```
predict('''I cannot seem to get along with my co-workers at work.
        This has affected my productivity and it gives me a lot of stress.
        I don't know how to resolve my situation.''')
```

```
Social Relationships
```

BERT Model

```
newdf = counseldf
```

```
newdf.topics = pd.Categorical(newdf.topics)
newdf['target'] = newdf.topics.cat.codes
```

Below dataset is divided into train and test sets.

```
msk = np.random.rand(len(newdf)) < 0.8

train_df = newdf[msk]

test_df = newdf[~msk]
```

```
train_df.head()
```

	questionText	topics	target
1	I'm planning to have baby, so I have to quit s...	Substance Abuse	29
3	I am extremely possessive in my relationships ...	Behavioral Change	4
5	I want [illegible]nship with someone that...	Relationship Dissolution	22
6	I easily recognize this but have no control ov...	Anger Management	2
7	It takes me a long time to fall asleep; I'd es...	Sleep Improvement	25

```
test_df.head()
```

	questionText	topics	target
0	My wife and mother are having tense disagreeme...	Family Conflict	12
2	I have secrets in my mind, and I don't know wh...	Family Conflict	12
4	I had a head injury a few years ago and my min...	Anxiety	3
10	Cheating is something unacceptable for me but ...	Relationships	23
11	I have a lot of issues going on right now. Fir...	Anxiety	3

The below code loads the BERT tokenizer, pre-trained BERT layer and prepares vocabulary file, and tokenizer object for the dataset.

```
import tensorflow_hub as hub
from tensorflow.keras import layers
import bert

#Loding pretrained bert layer
BertTokenizer = bert.bert_tokenization.FullTokenizer
bert_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1",
                            trainable=False)


# Loading tokenizer from the bert layer
vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()
bert_tokenizer = BertTokenizer(vocab_file, do_lower_case)


# function to encode the text into tokens, masks, and segment flags
import numpy as np
def bert_encode(texts, tokenizer, max_len=512):
    all_tokens = []
    all_masks = []
    all_segments = []

    for text in texts:
        text = tokenizer.tokenize(text)

        text = text[:max_len-2]
        input_sequence = ["[CLS]"] + text + ["[SEP]"]
        pad_len = max_len - len(input_sequence)

        tokens = tokenizer.convert_tokens_to_ids(input_sequence)
        tokens += [0] * pad_len
        pad_masks = [1] * len(input_sequence) + [0] * pad_len
        segment_ids = [0] * max_len

        all_tokens.append(tokens)
        all_masks.append(pad_masks)
        all_segments.append(segment_ids)

    return np.array(all_tokens), np.array(all_masks), np.array(all_segments)

MAX_LEN = 220

# encode train set
train_input = bert_encode(train_df.questionText.values, bert_tokenizer, max_len=MAX_LEN)
# encode  test set
test_input = bert_encode(test_df.questionText.values, bert_tokenizer, max_len= MAX_LEN )
train_labels = train_df.target.values


# from tf.keras.utils import to_categorical

categorical_labels = tf.keras.utils.to_categorical(train_labels, num_classes=32)
```

Finally, the BERT layer incorporated model is define below.

```
# first define input for token, mask and segment id
from tensorflow.keras.layers import  Input
input_word_ids = Input(shape=(MAX_LEN,), dtype=tf.int32, name="input_word_ids")
input_mask = Input(shape=(MAX_LEN,), dtype=tf.int32, name="input_mask")
segment_ids = Input(shape=(MAX_LEN,), dtype=tf.int32, name="segment_ids")

#  output
from tensorflow.keras.layers import Dense
pooled_output, sequence_output = bert_layer([input_word_ids, input_mask, segment_ids])
clf_output = sequence_output[:, 0, :]
out = Dense(32, activation='sigmoid')(clf_output)

# intilize model
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
BERT_model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=out)
BERT_model.compile(Adam(lr=2e-6), loss='categorical_crossentropy', metrics=['accuracy'])
BERT_model.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_word_ids (InputLayer)     [(None, 220)]        0
__________________________________________________________________________________________________
input_mask (InputLayer)         [(None, 220)]        0
__________________________________________________________________________________________________
segment_ids (InputLayer)        [(None, 220)]        0
__________________________________________________________________________________________________
keras_layer_1 (KerasLayer)      [(None, 768), (None, 109482241   input_word_ids[0][0]
                                                                 input_mask[0][0]
                                                                 segment_ids[0][0]
__________________________________________________________________________________________________
tf_op_layer_strided_slice (Tens [(None, 768)]        0           keras_layer_1[0][1]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 32)           24608       tf_op_layer_strided_slice[0][0]
==================================================================================================
Total params: 109,506,849
Trainable params: 24,608
Non-trainable params: 109,482,241
__________________________________________________________________________________________________
```

```
BERT_model.compile(optimizer = 'adam', loss='categorical_crossentropy', metrics=['accuracy', 'Recall', 'Precision'])
```

```
# train
train_history = BERT_model.fit(
    train_input, categorical_labels,
    validation_split=0.2,
    epochs=50,
    batch_size=32
)
```

```
Epoch 1/50
28/28 [==============================] - 277s 10s/step - loss: 1.6164 - accuracy: 0.5682 - recall: 0.0148 - precision: 0.8667 -
Epoch 2/50
28/28 [==============================] - 286s 10s/step - loss: 1.4952 - accuracy: 0.5716 - recall: 0.0136 - precision: 1.0000 -
Epoch 3/50
28/28 [==============================] - 292s 10s/step - loss: 1.4062 - accuracy: 0.6318 - recall: 0.0159 - precision: 1.0000 -
Epoch 4/50
28/28 [==============================] - 596s 21s/step - loss: 1.3343 - accuracy: 0.6375 - recall: 0.0114 - precision: 1.0000 -
Epoch 5/50
28/28 [==============================] - 299s 11s/step - loss: 1.2865 - accuracy: 0.6557 - recall: 0.0205 - precision: 1.0000 -
Epoch 6/50
28/28 [==============================] - 305s 11s/step - loss: 1.2235 - accuracy: 0.6875 - recall: 0.0114 - precision: 1.0000 -
Epoch 7/50
28/28 [==============================] - 303s 11s/step - loss: 1.1653 - accuracy: 0.7193 - recall: 0.0193 - precision: 1.0000 -
Epoch 8/50
28/28 [==============================] - 296s 11s/step - loss: 1.1215 - accuracy: 0.7182 - recall: 0.0170 - precision: 1.0000 -
Epoch 9/50
28/28 [==============================] - 326s 12s/step - loss: 1.0761 - accuracy: 0.7432 - recall: 0.0136 - precision: 1.0000 -
Epoch 10/50
28/28 [==============================] - 316s 11s/step - loss: 1.0358 - accuracy: 0.7534 - recall: 0.0170 - precision: 1.0000 -
Epoch 11/50
28/28 [==============================] - 320s 11s/step - loss: 1.0013 - accuracy: 0.7841 - recall: 0.0148 - precision: 1.0000 -
Epoch 12/50
28/28 [==============================] - 315s 11s/step - loss: 0.9731 - accuracy: 0.7750 - recall: 0.0193 - precision: 1.0000 -
Epoch 13/50
28/28 [==============================] - 317s 11s/step - loss: 0.9360 - accuracy: 0.7830 - recall: 0.0148 - precision: 1.0000 -
Epoch 14/50
28/28 [==============================] - 318s 11s/step - loss: 0.9059 - accuracy: 0.7989 - recall: 0.0216 - precision: 1.0000 -
Epoch 15/50
28/28 [==============================] - 318s 11s/step - loss: 0.8806 - accuracy: 0.8045 - recall: 0.0125 - precision: 1.0000 -
```

```
Epoch 16/50
28/28 [==============================] - 314s 11s/step - loss: 0.8504 - accuracy: 0.8148 - recall: 0.0148 - precision: 1.0000 -
Epoch 17/50
28/28 [==============================] - 265s 9s/step - loss: 0.8302 - accuracy: 0.8182 - recall: 0.0148 - precision: 1.0000 - v
Epoch 18/50
28/28 [==============================] - 262s 9s/step - loss: 0.8072 - accuracy: 0.8227 - recall: 0.0148 - precision: 1.0000 - v
Epoch 19/50
28/28 [==============================] - 260s 9s/step - loss: 0.7856 - accuracy: 0.8386 - recall: 0.0148 - precision: 1.0000 - v
Epoch 20/50
28/28 [==============================] - 260s 9s/step - loss: 0.7656 - accuracy: 0.8489 - recall: 0.0148 - precision: 1.0000 - v
Epoch 21/50
28/28 [==============================] - 259s 9s/step - loss: 0.7456 - accuracy: 0.8545 - recall: 0.0148 - precision: 1.0000 - v
Epoch 22/50
28/28 [==============================] - 257s 9s/step - loss: 0.7269 - accuracy: 0.8477 - recall: 0.0148 - precision: 1.0000 - v
Epoch 23/50
28/28 [==============================] - 254s 9s/step - loss: 0.7141 - accuracy: 0.8523 - recall: 0.0148 - precision: 1.0000 - v
Epoch 24/50
28/28 [==============================] - 250s 9s/step - loss: 0.6854 - accuracy: 0.8693 - recall: 0.0148 - precision: 1.0000 - v
Epoch 25/50
28/28 [==============================] - 251s 9s/step - loss: 0.6762 - accuracy: 0.8648 - recall: 0.0148 - precision: 1.0000 - v
Epoch 26/50
28/28 [==============================] - 254s 9s/step - loss: 0.6573 - accuracy: 0.8750 - recall: 0.0148 - precision: 1.0000 - v
Epoch 27/50
28/28 [==============================] - 254s 9s/step - loss: 0.6372 - accuracy: 0.8761 - recall: 0.0148 - precision: 1.0000 - v
Epoch 28/50
28/28 [==============================] - 254s 9s/step - loss: 0.6267 - accuracy: 0.8784 - recall: 0.0148 - precision: 1.0000 - v
```

Add text cell

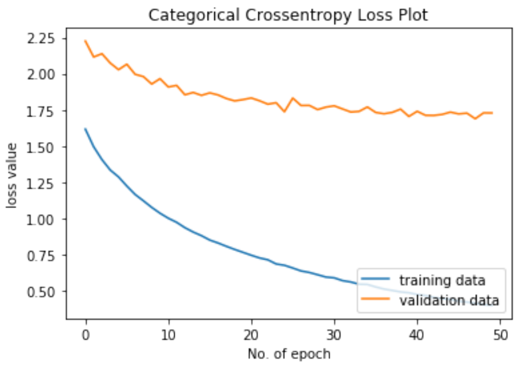
```
BERT_model.save('BERT_model.h5')
```

```
train_history_df = pd.DataFrame(train_history.history)
train_history_df['f1_score'] = (2 * train_history_df['precision']*train_history_df['recall'])/(train_history_df['precision']+train_histo
train_history_df['val_f1_score'] = (2 * train_history_df['val_precision']*train_history_df['val_recall'])/(train_history_df['val_precis:
```

```
history_df.head(30)
```

	loss	accuracy	precision	recall	val_loss	val_accuracy	val_precision	val_recall	f1_score	val_f1_score
0	3.428652	0.194185	0.000000	0.000000	3.323464	0.225182	0.000000	0.000000	NaN	NaN
1	3.117190	0.205607	0.324324	0.012461	2.845594	0.205811	0.000000	0.000000	0.024000	NaN

```
plt.plot(train_ Add text cell :ory['loss'], label='training data')
plt.plot(train_history.history['val_loss'], label='validation data')
plt.title('Categorical Crossentropy Loss Plot')
plt.ylabel('loss value')
plt.xlabel('No. of epoch')
plt.legend(loc="lower right")
plt.show()
```



16	0.122799	0.978193	0.998899	0.941848	2.432647	0.716707	0.767956	0.673123	0.969535	0.717419

```
plt.subplots(figsize=(15,10))
ax = plt.subplot(2,2,1)
plt.plot(train_history.history['accuracy'], label='training_accuracy')
plt.plot(train_history.history['val_accuracy'], label='test_accuracy')
plt.title('Accuracy Plot')
plt.ylabel('accuracy value')
plt.xlabel('No. of epoch')
plt.legend(loc="lower right")

ax = plt.subplot(2,2,2)
plt.plot(train_history.history['precision'], label='training_precision')
plt.plot(train_history.history['val_precision'], label='test_precision')
plt.title('Precision Plot')
plt.ylabel('precision value')
plt.xlabel('No. of epoch')
plt.legend(loc="lower right")

ax = plt.subplot(2,2,3)
plt.plot(train_history.history['recall'], label='training_recall')
plt.plot(train_history.history['val_recall'], label='test_recall')
```