```
#import file
from google.colab import files
uploaded = files.upload()
```

Choose Files Corona_NLP_test.csv

- **Corona_NLP_test.csv**(text/csv) - 1002494 bytes, last modified: 4/21/2023 - 100% done

```
Saving Corona_NLP_test.csv to Corona_NLP_test.csv
```

```
import io
import pandas as pd
```

```
#preprocess data
df = pd.read_csv(io.BytesIO(uploaded['Corona_NLP_test.csv']))
df1 = df.drop(['UserName', 'ScreenName', 'Location', 'TweetAt'], axis = 1)
df1.Sentiment.replace(('Extremely Positive','Positive', 'Negative', 'Extremely Negative', 'Neutral'), ('positive', 'positive', 'negative', 'n
df1
```

	OriginalTweet	Sentiment
0	TRENDING: New Yorkers encounter empty supermar...	negative
1	When I couldn't find hand sanitizer at Fred Me...	positive
2	Find out how you can protect yourself and love...	positive
3	#Panic buying hits #NewYork City as anxious sh...	negative
4	#toiletpaper #dunnypaper #coronavirus #coronav...	positive
...	...	...
3793	Meanwhile In A Supermarket in Israel -- People...	positive
3794	Did you panic buy a lot of non-perishable item...	negative
3795	Asst Prof of Economics @cconces was on @NBCPhi...	positive
3796	Gov need to do somethings instead of biar je r...	negative
3797	I and @ForestandPaper members are committed to...	positive

3798 rows × 2 columns

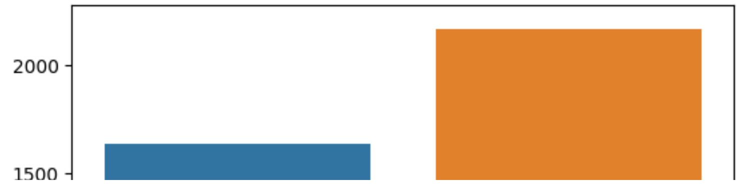
```
df1.head()
```

	OriginalTweet	Sentiment
0	TRENDING: New Yorkers encounter empty supermar...	negative
1	When I couldn't find hand sanitizer at Fred Me...	positive
2	Find out how you can protect yourself and love...	positive
3	#Panic buying hits #NewYork City as anxious sh...	negative
4	#toiletpaper #dunnypaper #coronavirus #coronav...	positive

```
#data visualization
import seaborn as sns

sns.countplot(data = df1, x = 'Sentiment')
```

```
<Axes: xlabel='Sentiment', ylabel='count'>
```



Here we can see the distribution of the data from the dataset that was chosen. I chose a dataset from kaggle. https://www.kaggle.com/datasets/datatattle/covid-19-nlp-text-classification?resource=download. This dataset looks at COVID related tweets and performs sentiment analysis on them. Here as we can see, there are a larger amount of positive tweets within the dataset but not enough to skew it in any sort of way

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import pickle
import numpy as np
import pandas as pd
np.random.seed(1234)
```

```
print('rows and columns:', df.shape)
```

```
     rows and columns: (3798, 6)
```

```
#split dataset into train and test samples
i = np.random.rand(len(df)) < 0.8
train = df1[i]
test = df1[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
     train data size:  (3027, 2)
     test data size:  (771, 2)
```

```
#set up X and Y train and test
num_labels = 2
vocab_size = 25000
batch_size = 100

tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.OriginalTweet)

x_train = tokenizer.texts_to_matrix(train.OriginalTweet, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.OriginalTweet, mode='tfidf')
```

```
#set up X and Y train and test
encoder = LabelEncoder()
encoder.fit(train.Sentiment)
y_train = encoder.transform(train.Sentiment)
y_test = encoder.transform(test.Sentiment)
```

```
print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
print("test first five labels:", y_test[:5])
```

```
     train shapes: (3027, 25000) (3027,)
     test shapes: (771, 25000) (771,)
     test first five labels: [1 0 1 1 0]
```

```
# Sequential model fit
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal', activation='relu'))
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1)
```

```
Epoch 2/30
28/28 [==============================] - 1s 20ms/step - loss: 0.5718 - accuracy: 0.7419 - val_loss: 0.6143 - val_accuracy: 0.6931
Epoch 3/30
28/28 [==============================] - 1s 18ms/step - loss: 0.3922 - accuracy: 0.9214 - val_loss: 0.5698 - val_accuracy: 0.7129
Epoch 4/30
28/28 [==============================] - 1s 19ms/step - loss: 0.2181 - accuracy: 0.9666 - val_loss: 0.5619 - val_accuracy: 0.7294
Epoch 5/30
28/28 [==============================] - 0s 13ms/step - loss: 0.1163 - accuracy: 0.9890 - val_loss: 0.5718 - val_accuracy: 0.7360
Epoch 6/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0668 - accuracy: 0.9971 - val_loss: 0.5828 - val_accuracy: 0.7360
Epoch 7/30
28/28 [==============================] - 0s 13ms/step - loss: 0.0421 - accuracy: 0.9993 - val_loss: 0.5964 - val_accuracy: 0.7327
Epoch 8/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0286 - accuracy: 0.9996 - val_loss: 0.6127 - val_accuracy: 0.7393
Epoch 9/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0206 - accuracy: 1.0000 - val_loss: 0.6292 - val_accuracy: 0.7393
Epoch 10/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0156 - accuracy: 1.0000 - val_loss: 0.6455 - val_accuracy: 0.7393
Epoch 11/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0121 - accuracy: 1.0000 - val_loss: 0.6589 - val_accuracy: 0.7360
Epoch 12/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0097 - accuracy: 1.0000 - val_loss: 0.6733 - val_accuracy: 0.7393
Epoch 13/30
28/28 [==============================] - 0s 13ms/step - loss: 0.0079 - accuracy: 1.0000 - val_loss: 0.6862 - val_accuracy: 0.7360
Epoch 14/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0065 - accuracy: 1.0000 - val_loss: 0.6979 - val_accuracy: 0.7360
Epoch 15/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0055 - accuracy: 1.0000 - val_loss: 0.7089 - val_accuracy: 0.7327
Epoch 16/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.7195 - val_accuracy: 0.7360
Epoch 17/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0041 - accuracy: 1.0000 - val_loss: 0.7290 - val_accuracy: 0.7294
Epoch 18/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.7393 - val_accuracy: 0.7327
Epoch 19/30
28/28 [==============================] - 0s 13ms/step - loss: 0.0031 - accuracy: 1.0000 - val_loss: 0.7497 - val_accuracy: 0.7327
Epoch 20/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0028 - accuracy: 1.0000 - val_loss: 0.7590 - val_accuracy: 0.7327
Epoch 21/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0025 - accuracy: 1.0000 - val_loss: 0.7670 - val_accuracy: 0.7360
Epoch 22/30
28/28 [==============================] - 0s 13ms/step - loss: 0.0022 - accuracy: 1.0000 - val_loss: 0.7755 - val_accuracy: 0.7327
Epoch 23/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.7831 - val_accuracy: 0.7327
Epoch 24/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.7909 - val_accuracy: 0.7360
Epoch 25/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.7983 - val_accuracy: 0.7327
Epoch 26/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.8057 - val_accuracy: 0.7327
Epoch 27/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.8124 - val_accuracy: 0.7327
Epoch 28/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.8186 - val_accuracy: 0.7327
Epoch 29/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.8255 - val_accuracy: 0.7327
Epoch 30/30
28/28 [==============================] - 0s 12ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.8313 - val_accuracy: 0.7327
```

```
#evaluation
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
```

```
8/8 [==============================] - 0s 4ms/step - loss: 0.9157 - accuracy: 0.6952
Accuracy:  0.69520103931427
```

```
print(score)
```

```
[0.9156782031059265, 0.69520103931427]
```

```
pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]
```

```
25/25 [==============================] - 0s 2ms/step
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
     accuracy score:  0.695201037613489
     precision score:  0.706140350877193
     recall score:  0.7612293144208038
     f1 score:  0.732650739476678
```

As we can see here, using a sequential model in terms of this selected dataset, we have reached an accuracy score of 0.6952. A precision of 0.706, recall of 0.76123, and f1 score of 0.7327. As a result, we can see that this model is pretty accurate as a whole.

```
from tensorflow.keras import layers, models, preprocessing

max_features = 10000
maxlen = 500
batch_size = 32
```

```
#split dataset into train and test
i = np.random.rand(len(df)) < 0.8
train = df1[i]
test = df1[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
     train data size:  (3040, 2)
     test data size:  (758, 2)
```

```
#set up X and Y train and test
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.OriginalTweet)

x_train = tokenizer.texts_to_matrix(train.OriginalTweet, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.OriginalTweet, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.Sentiment)
y_train = encoder.transform(train.Sentiment)
y_test = encoder.transform(test.Sentiment)

train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
train_data.shape
```

```
     (3040, 500)
```

```
#build CNN network
model = models.Sequential()
model.add(layers.Embedding(max_features, 128, input_length=maxlen))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))
```

```
model.summary()
```

```
     Model: "sequential_1"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #   
     =================================================================
      embedding (Embedding)       (None, 500, 128)          1280000   
                                                                      
      conv1d (Conv1D)             (None, 494, 32)           28704     
                                                                      
      max_pooling1d (MaxPooling1D  (None, 98, 32)           0         
      )                                                               
                                                                      
      conv1d_1 (Conv1D)           (None, 92, 32)            7200      
                                                                      
      global_max_pooling1d (Globa  (None, 32)               0         
      lMaxPooling1D)

      dense_2 (Dense)             (None, 1)                 33

     =================================================================
     Total params: 1,315,937
     Trainable params: 1,315,937
     Non-trainable params: 0
     _________________________________________________________________
```

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),  # set learning rate
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
19/19 [==============================] - 7s 349ms/step - loss: 1.4804 - accuracy: 0.4408 - val_loss: 1.2235 - val_accuracy: 0.3964
Epoch 2/10
19/19 [==============================] - 6s 296ms/step - loss: 0.9981 - accuracy: 0.4408 - val_loss: 0.9064 - val_accuracy: 0.3964
Epoch 3/10
19/19 [==============================] - 6s 339ms/step - loss: 0.7816 - accuracy: 0.4408 - val_loss: 0.7249 - val_accuracy: 0.3964
Epoch 4/10
19/19 [==============================] - 6s 297ms/step - loss: 0.6949 - accuracy: 0.5090 - val_loss: 0.6761 - val_accuracy: 0.6036
Epoch 5/10
19/19 [==============================] - 6s 336ms/step - loss: 0.6866 - accuracy: 0.5592 - val_loss: 0.6740 - val_accuracy: 0.6036
Epoch 6/10
19/19 [==============================] - 6s 297ms/step - loss: 0.6864 - accuracy: 0.5592 - val_loss: 0.6865 - val_accuracy: 0.6036
Epoch 7/10
19/19 [==============================] - 6s 337ms/step - loss: 0.6875 - accuracy: 0.5592 - val_loss: 0.6733 - val_accuracy: 0.6036
Epoch 8/10
19/19 [==============================] - 6s 294ms/step - loss: 0.6867 - accuracy: 0.5592 - val_loss: 0.6742 - val_accuracy: 0.6036
Epoch 9/10
19/19 [==============================] - 6s 338ms/step - loss: 0.6874 - accuracy: 0.5592 - val_loss: 0.6750 - val_accuracy: 0.6036
Epoch 10/10
19/19 [==============================] - 6s 295ms/step - loss: 0.6874 - accuracy: 0.5592 - val_loss: 0.6779 - val_accuracy: 0.6036
```

```
from sklearn import metrics

pred = model.predict(test_data)
pred = [1.0 if p>= 0.01 else 0 for p in pred]
```

```
24/24 [==============================] - 0s 2ms/step
```

```
print(metrics.f1_score(y_test, pred, average='weighted'))
print(metrics.precision_score(y_test, pred, average='weighted'))
print(metrics.recall_score(y_test, pred, average='weighted'))
```

```
0.3936873932511656
0.30567428123919l5
0.5528781793842035
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and bei
  _warn_prf(average, modifier, msg_start, len(result))
```

Here, using the CNN architecture, we get an f1 score of 0.3937, precision score of 0.3057, and recall score 0.5529

```
#split dataset into train and test
max_features = 10000
maxlen = 500
batch_size = 32

i = np.random.rand(len(df)) < 0.8
train = df1[i]
test = df1[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
train data size:  (3056, 2)
test data size:  (742, 2)
```

```
#set up X and Y train and test
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.OriginalTweet)

x_train = tokenizer.texts_to_matrix(train.OriginalTweet, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.OriginalTweet, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.Sentiment)
y_train = encoder.transform(train.Sentiment)
y_test = encoder.transform(test.Sentiment)

train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)


#build RNN network architecture
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.SimpleRNN(32))
model.add(layers.Dense(1, activation='sigmoid'))


model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, None, 32)          320000

 simple_rnn (SimpleRNN)      (None, 32)                2080

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])


history = model.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
20/20 [==============================] - 3s 98ms/step - loss: 0.6866 - accuracy: 0.5615 - val_loss: 0.6686 - val_accuracy: 0.6105
Epoch 2/10
20/20 [==============================] - 2s 89ms/step - loss: 0.6856 - accuracy: 0.5656 - val_loss: 0.6691 - val_accuracy: 0.6105
Epoch 3/10
20/20 [==============================] - 2s 89ms/step - loss: 0.6852 - accuracy: 0.5656 - val_loss: 0.6712 - val_accuracy: 0.6105
Epoch 4/10
20/20 [==============================] - 2s 89ms/step - loss: 0.6855 - accuracy: 0.5656 - val_loss: 0.6760 - val_accuracy: 0.6105
Epoch 5/10
20/20 [==============================] - 2s 88ms/step - loss: 0.6848 - accuracy: 0.5656 - val_loss: 0.6795 - val_accuracy: 0.6105
Epoch 6/10
20/20 [==============================] - 3s 132ms/step - loss: 0.6857 - accuracy: 0.5656 - val_loss: 0.6704 - val_accuracy: 0.6105
Epoch 7/10
20/20 [==============================] - 2s 87ms/step - loss: 0.6849 - accuracy: 0.5656 - val_loss: 0.6739 - val_accuracy: 0.6105
Epoch 8/10
20/20 [==============================] - 2s 88ms/step - loss: 0.6855 - accuracy: 0.5656 - val_loss: 0.6707 - val_accuracy: 0.6105
Epoch 9/10
20/20 [==============================] - 2s 89ms/step - loss: 0.6851 - accuracy: 0.5656 - val_loss: 0.6703 - val_accuracy: 0.6105
Epoch 10/10
20/20 [==============================] - 2s 90ms/step - loss: 0.6853 - accuracy: 0.5656 - val_loss: 0.6706 - val_accuracy: 0.6105
```

```
print('f1 Score: ', metrics.f1_score(y_test, pred, average='weighted'))
print('Precision Score: ', metrics.precision_score(y_test, pred, average='weighted'))
print('Recall Score: ', metrics.recall_score(y_test, pred, average='weighted'))
```

```
f1 Score:  0.3936873932511656
Precision Score:  0.3056742812391915
Recall Score:  0.5528781793842035
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and bei
  _warn_prf(average, modifier, msg_start, len(result))
```

Using a RNN network, we have an f1 score of 0.3937, precision score of 0.3057, and recall score of 0.5529

```
max_features = 10000
maxlen = 20
batch_size = 32

i = np.random.rand(len(df)) < 0.8
train = df1[i]
test = df1[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
train data size:  (3051, 2)
test data size:  (747, 2)
```

```
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.OriginalTweet)

x_train = tokenizer.texts_to_matrix(train.OriginalTweet, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.OriginalTweet, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.Sentiment)
y_train = encoder.transform(train.Sentiment)
y_test = encoder.transform(test.Sentiment)

train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)


model = models.Sequential()
model.add(layers.Embedding(max_features, 8, input_length=maxlen))
model.add(layers.Flatten())
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(train_data, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_7 (Embedding)     (None, 20, 8)             80000     
                                                                 
 flatten_4 (Flatten)         (None, 160)               0         
                                                                 
 dense_11 (Dense)            (None, 16)                2576      
                                                                 
 dense_12 (Dense)            (None, 1)                 17        
                                                                 
=================================================================
Total params: 82,593
Trainable params: 82,593
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
77/77 [==============================] - 1s 3ms/step - loss: 0.6876 - acc: 0.5545 - val_loss: 0.6704 - val_acc: 0.6236
Epoch 2/10
77/77 [==============================] - 0s 2ms/step - loss: 0.6859 - acc: 0.5619 - val_loss: 0.6712 - val_acc: 0.6236
Epoch 3/10
77/77 [==============================] - 0s 2ms/step - loss: 0.6858 - acc: 0.5619 - val_loss: 0.6701 - val_acc: 0.6236
Epoch 4/10
77/77 [==============================] - 0s 2ms/step - loss: 0.6855 - acc: 0.5619 - val_loss: 0.6666 - val_acc: 0.6236
Epoch 5/10
77/77 [==============================] - 0s 2ms/step - loss: 0.6859 - acc: 0.5619 - val_loss: 0.6725 - val_acc: 0.6236
Epoch 6/10
77/77 [==============================] - 0s 2ms/step - loss: 0.6859 - acc: 0.5619 - val_loss: 0.6701 - val_acc: 0.6236
Epoch 7/10
77/77 [==============================] - 0s 3ms/step - loss: 0.6858 - acc: 0.5619 - val_loss: 0.6708 - val_acc: 0.6236
Epoch 8/10
77/77 [==============================] - 0s 2ms/step - loss: 0.6858 - acc: 0.5619 - val_loss: 0.6681 - val_acc: 0.6236
```

```
Epoch 9/10
77/77 [==============================] - 0s 2ms/step - loss: 0.6859 - acc: 0.5619 - val_loss: 0.6720 - val_acc: 0.6236
Epoch 10/10
77/77 [==============================] - 0s 2ms/step - loss: 0.6859 - acc: 0.5619 - val_loss: 0.6730 - val_acc: 0.6236
```

```
print('f1 Score: ', metrics.f1_score(y_test, pred, average='weighted'))
print('Precision Score: ', metrics.precision_score(y_test, pred, average='weighted'))
print('Recall Score: ', metrics.recall_score(y_test, pred, average='weighted'))
```

```
f1 Score:  0.3936873932511656
Precision Score:  0.3056742812391915
Recall Score:  0.5528781793842035
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and bei
  _warn_prf(average, modifier, msg_start, len(result))
```

Using an embedded approach, we have an f1 score of 0.3937, precision score of 0.3057, and recall score of 0.5529

! 32s completed at 11:10 PM