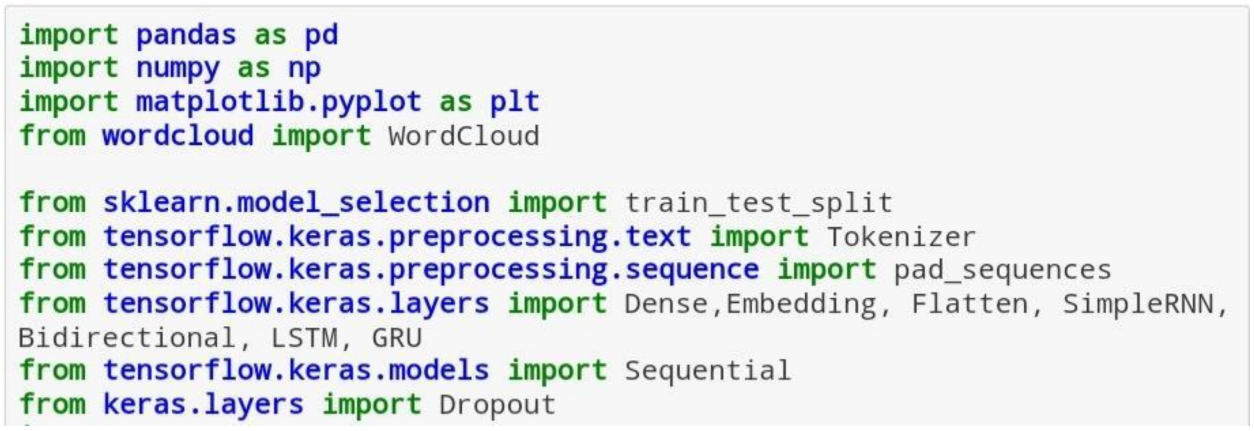
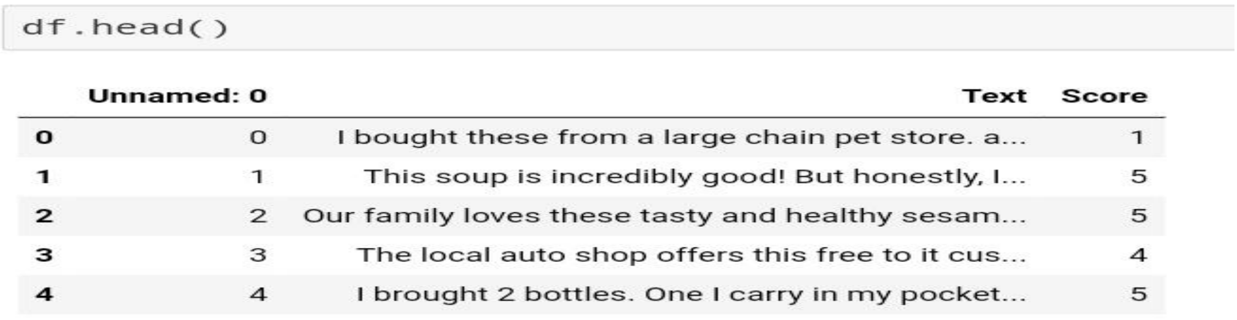
Project - Amazon food review sentiment analysis.

- Imports



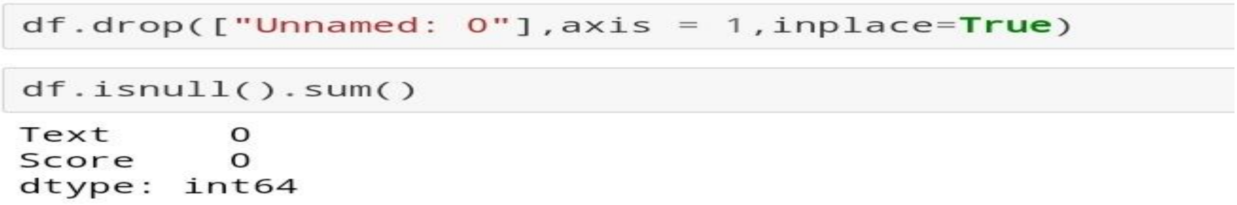
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud

from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense,Embedding, Flatten, SimpleRNN,
Bidirectional, LSTM, GRU
from tensorflow.keras.models import Sequential
from keras.layers import Dropout
```

- Read data and display df.head() to view starting columns of data.



```
df.head()
```

	Unnamed: 0	Text	Score
0	0	I bought these from a large chain pet store. a...	1
1	1	This soup is incredibly good! But honestly, I...	5
2	2	Our family loves these tasty and healthy sesam...	5
3	3	The local auto shop offers this free to it cus...	4
4	4	I brought 2 bottles. One I carry in my pocket...	5



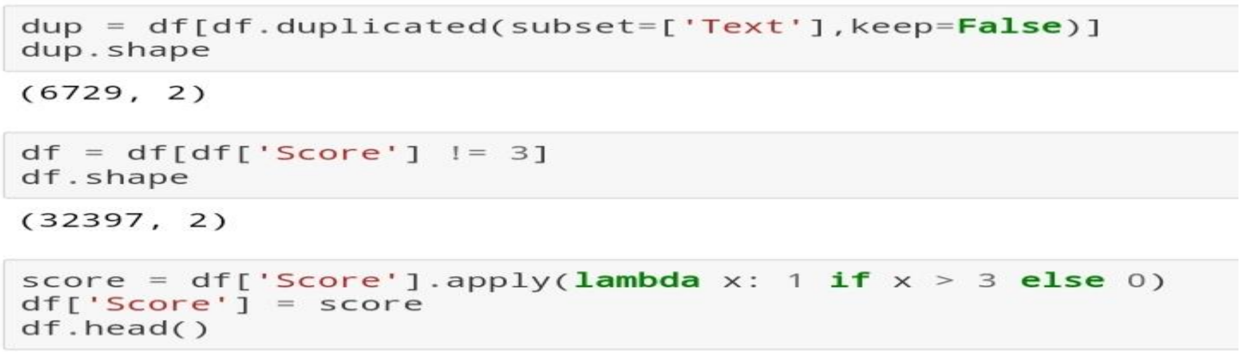
```
df.drop(["Unnamed: 0"],axis = 1,inplace=True)
```

```
df.isnull().sum()
```

```
Text      0
Score     0
dtype:  int64
```

- Remove Unwanted column from data and check if there is any null value present in data or not.

Remove duplicate from Text column which is repeating & in the score column if Score is > 3 then replace it with 1 which is positive score and if score is <= 3 then replace it with 0 which negative score.



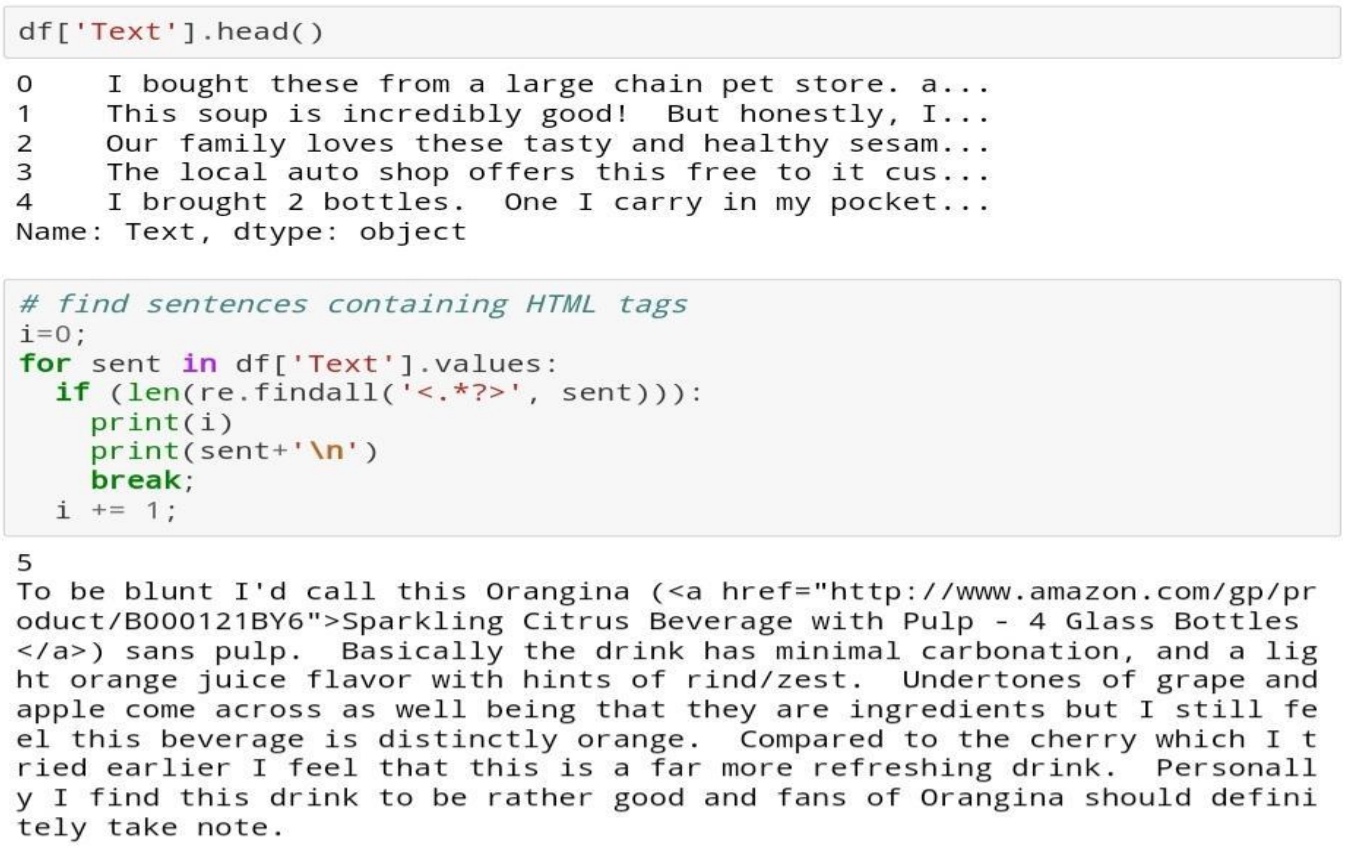
```
dup = df[df.duplicated(subset=['Text'],keep=False)]
dup.shape
```

```
(6729, 2)
```

```
df = df[df['Score'] != 3]
df.shape
```

```
(32397, 2)
```

```
score = df['Score'].apply(lambda x: 1 if x > 3 else 0)
df['Score'] = score
df.head()
```



```
df['Text'].head()
```

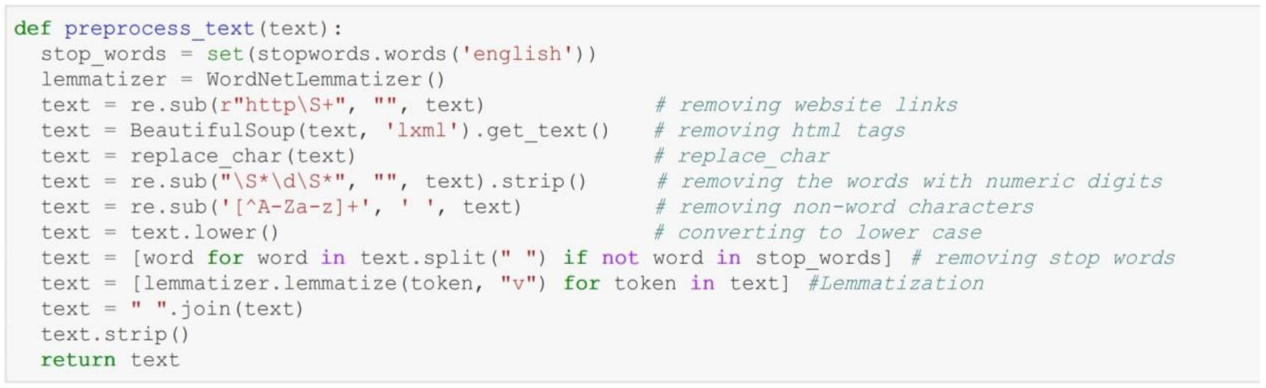
```
0    I bought these from a large chain pet store. a...
1    This soup is incredibly good!  But honestly, I...
2    Our family loves these tasty and healthy sesam...
3    The local auto shop offers this free to it cus...
4    I brought 2 bottles.  One I carry in my pocket...
Name: Text, dtype: object
```

```
# find sentences containing HTML tags
i=0;
for sent in df['Text'].values:
  if (len(re.findall('<.*?>', sent))):
    print(i)
    print(sent+'\n')
    break;
  i += 1;
```

```
5
To be blunt I'd call this Orangina (<a href="http://www.amazon.com/gp/pr
oduct/B000121BY6">Sparkling Citrus Beverage with Pulp - 4 Glass Bottles
</a>) sans pulp.  Basically the drink has minimal carbonation, and a lig
ht orange juice flavor with hints of rind/zest.  Undertones of grape and
apple come across as well being that they are ingredients but I still fe
el this beverage is distinctly orange.  Compared to the cherry which I t
ried earlier I feel that this is a far more refreshing drink.  Personall
y I find this drink to be rather good and fans of Orangina should defini
tely take note.
```

- Check Text column we can observe there are some stopwords present as well as we look if there is any HTML tag or any website link present on data or not and we find some tags with a website link.

Created a function.



```
def preprocess_text(text):
  stop_words = set(stopwords.words('english'))
  lemmatizer = WordNetLemmatizer()
  text = re.sub(r"http\S+", "", text)              # removing website links
  text = BeautifulSoup(text, 'lxml').get_text()    # removing html tags
  text = replace_char(text)                        # replace_char
  text = re.sub("\S*\d\S*", "", text).strip()      # removing the words with numeric digits
  text = re.sub('[^A-Za-z]+', ' ', text)           # removing non-word characters
  text = text.lower()                              # converting to lower case
  text = [word for word in text.split(" ") if not word in stop_words] # removing stop words
  text = [lemmatizer.lemmatize(token, "v") for token in text] #Lemmatization
  text = " ".join(text)
  text.strip()
  return text
```

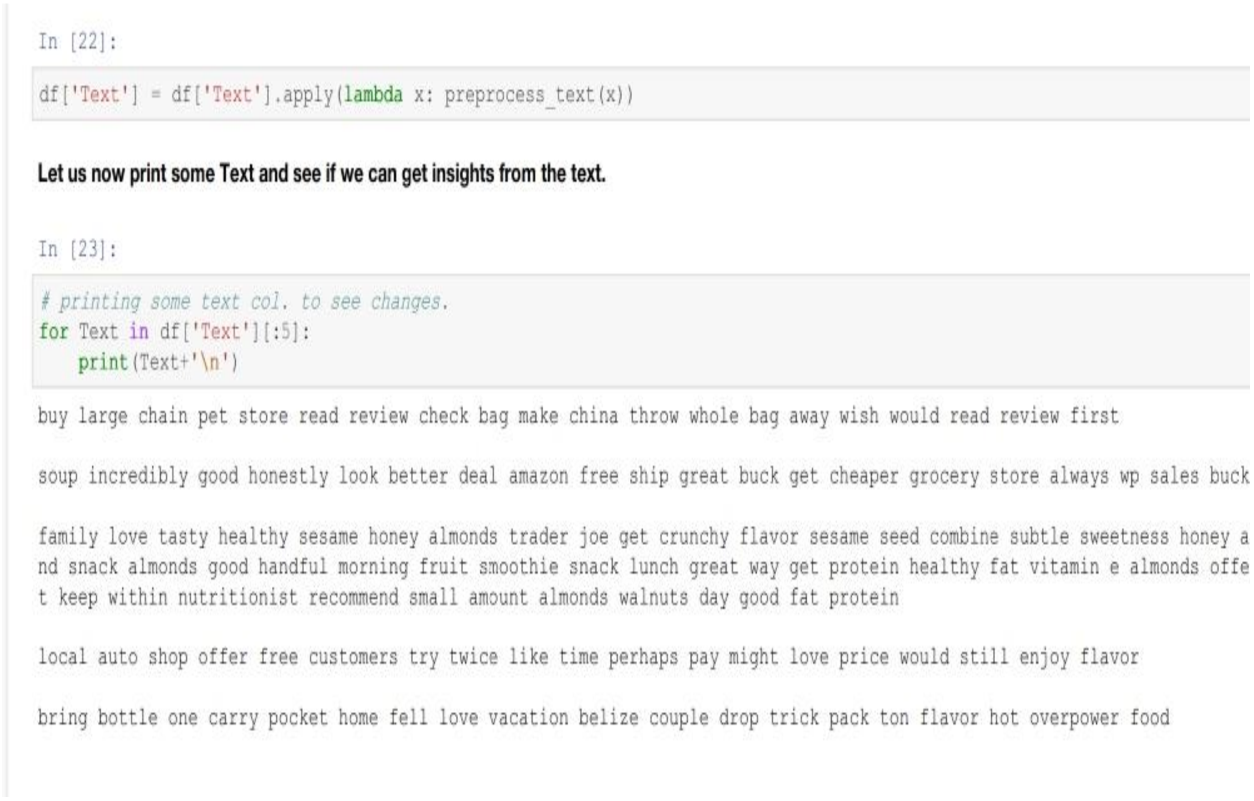
Above function will be used on text column to remove all the html tags, website links, removing stop words, to convert text to lower case, lemmatization, etc.

Used **RegEx**

Regular expression or RegEx in Python is denoted as RE are imported through re module. Python supports regular expression through libraries. RegEx in Python supports various things like Modifiers, Identifiers, and White space characters.

re.sub() - The re. sub() function is used to replace occurrences of a particular sub-string with another sub-string.

We apply function on text column and call head function to check if stopwords, etc got our result or not and the function works in above we find there are some words I, am, etc are removed.



In [22]:

```
df['Text'] = df['Text'].apply(lambda x: preprocess_text(x))
```

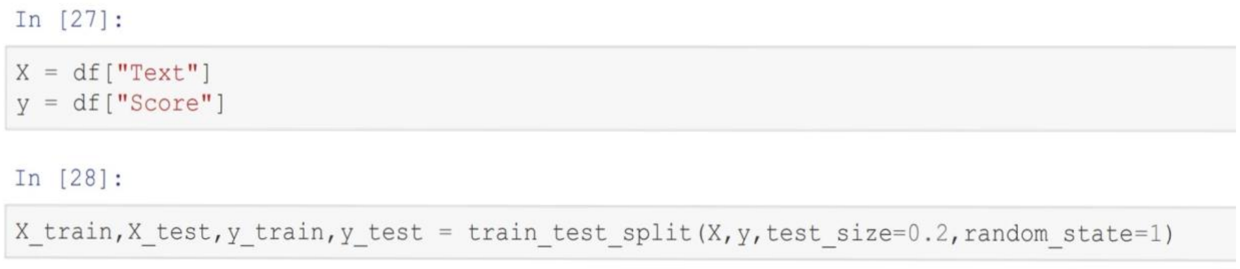
Let us now print some Text and see if we can get insights from the text.

In [23]:

```
# printing some text col. to see changes.
for Text in df['Text'][:5]:
    print(Text+'\n')
```

```
buy large chain pet store read review check bag make china throw whole bag away wish would read review first

soup incredibly good honestly look better deal amazon free ship great buck get cheaper grocery store always wp sales buck

family love tasty healthy sesame honey almonds trader joe get crunchy flavor sesame seed combine subtle sweetness honey a
nd snack almonds good handful morning fruit smoothie snack lunch great way get protein healthy fat vitamin e almonds offe
t keep within nutritionist recommend small amount almonds walnuts day good fat protein

local auto shop offer free customers try twice like time perhaps pay might love price would still enjoy flavor

bring bottle one carry pocket home fell love vacation belize couple drop trick pack ton flavor hot overpower food
```

Splitting the target and feature column and splitting data into training and testing.



In [27]:

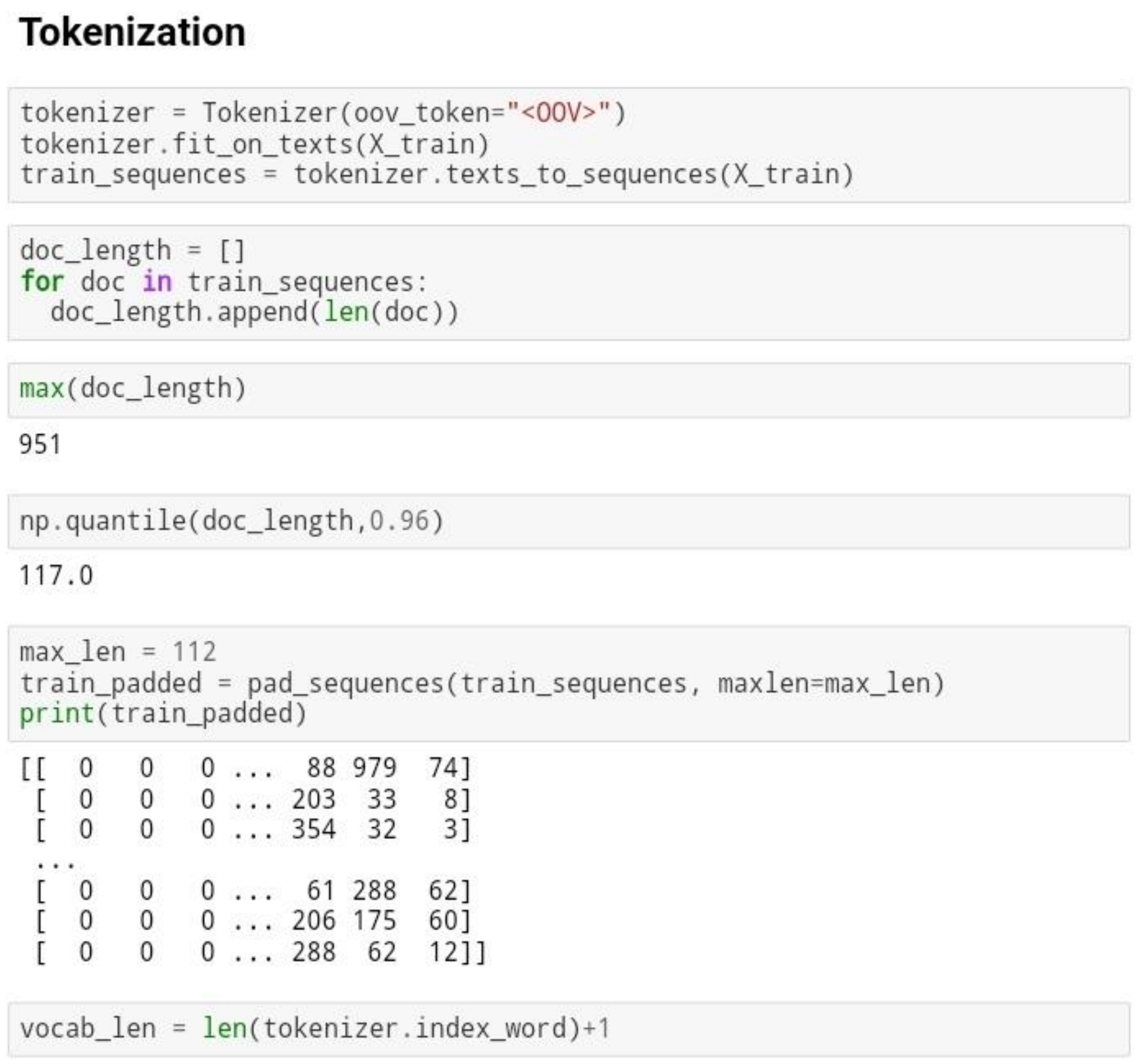
```
X = df["Text"]
y = df["Score"]
```

In [28]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

Applying Tokenization on training data.

Tokenization is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.



Tokenization

```
tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)
train_sequences = tokenizer.texts_to_sequences(X_train)
```

```
doc_length = []
for doc in train_sequences:
  doc_length.append(len(doc))
```

```
max(doc_length)
```

```
951
```

```
np.quantile(doc_length,0.96)
```

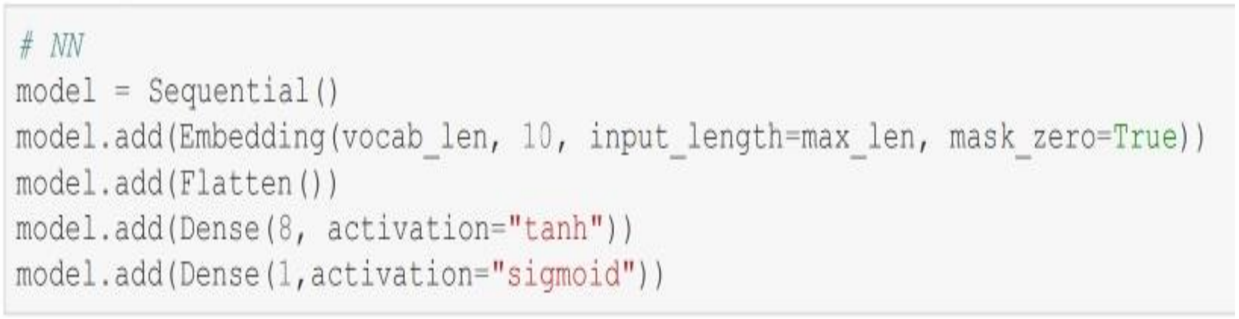
```
117.0
```

```
max_len = 112
train_padded = pad_sequences(train_sequences, maxlen=max_len)
print(train_padded)
```

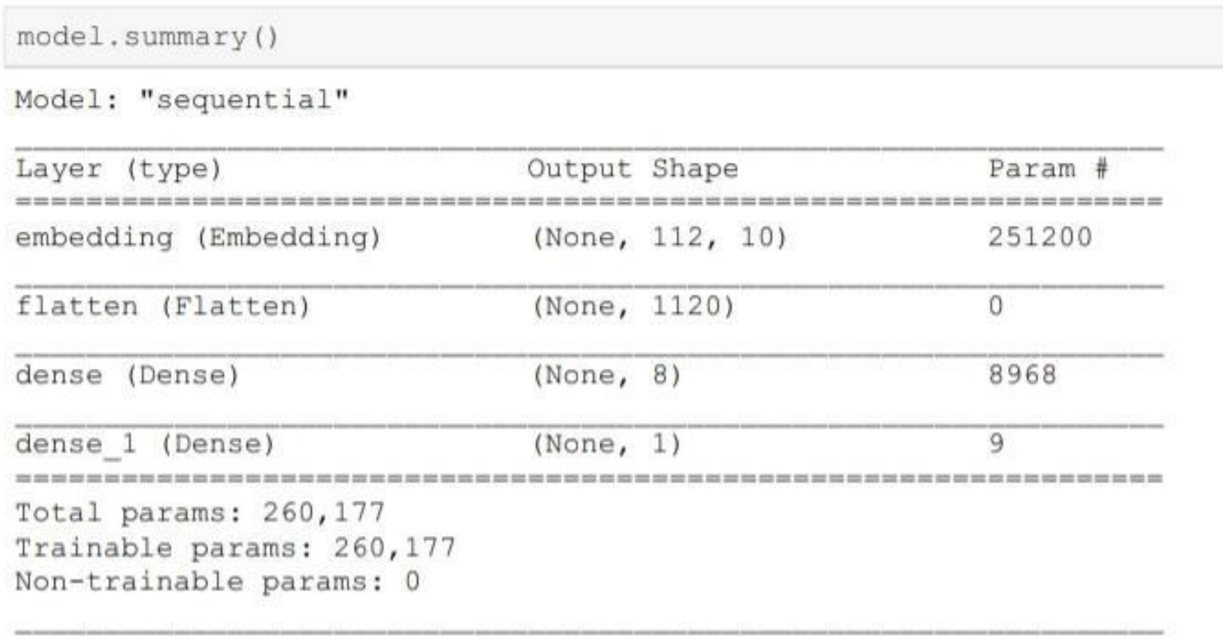
```
[[  0   0   0 ...  88 979  74]
 [  0   0   0 ... 203  33   8]
 [  0   0   0 ... 354  32   3]
 ...
 [  0   0   0 ...  61 288  62]
 [  0   0   0 ... 206 175  60]
 [  0   0   0 ... 288  62  12]]
```

```
vocab_len = len(tokenizer.index_word)+1
```

➢ **Neural network**.



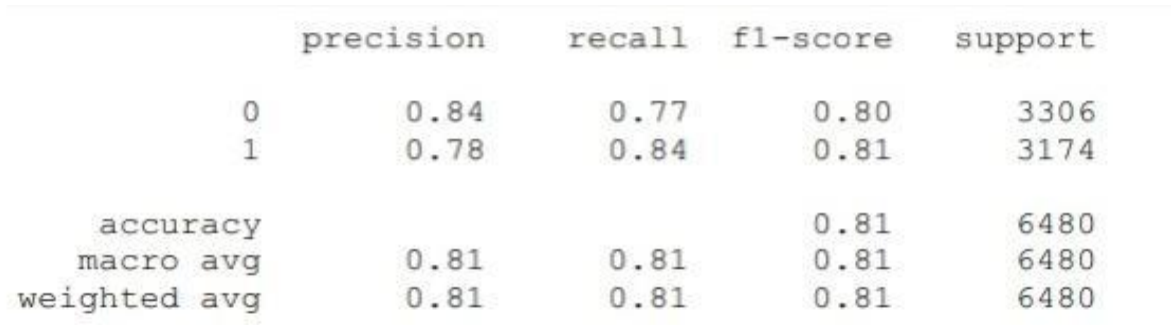
```
# NN
model = Sequential()
model.add(Embedding(vocab_len, 10, input_length=max_len, mask_zero=True))
model.add(Flatten())
model.add(Dense(8, activation="tanh"))
model.add(Dense(1,activation="sigmoid"))
```



```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 112, 10)           251200
_________________________________________________________________
flatten (Flatten)            (None, 1120)              0
_________________________________________________________________
dense (Dense)                (None, 8)                 8968
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 9
=================================================================
Total params: 260,177
Trainable params: 260,177
Non-trainable params: 0
_________________________________________________________________
```

Neural network testing model with accuracy 81.

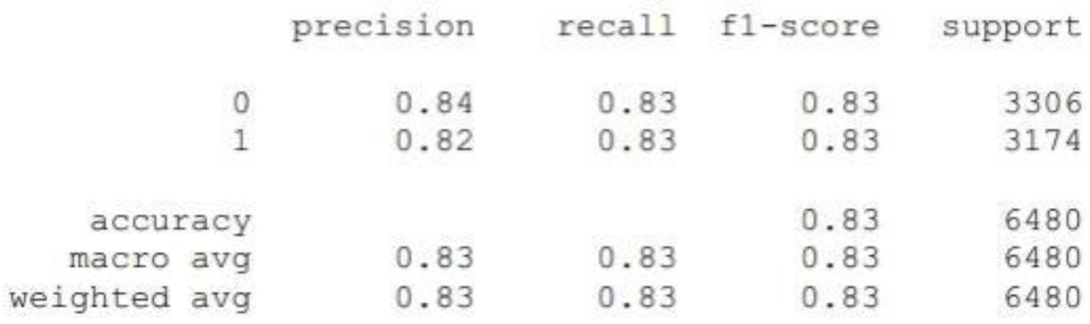


```
              precision    recall  f1-score   support

           0       0.84      0.77      0.80      3306
           1       0.78      0.84      0.81      3174

    accuracy                           0.81      6480
   macro avg       0.81      0.81      0.81      6480
weighted avg       0.81      0.81      0.81      6480
```

Will try to improve the score…

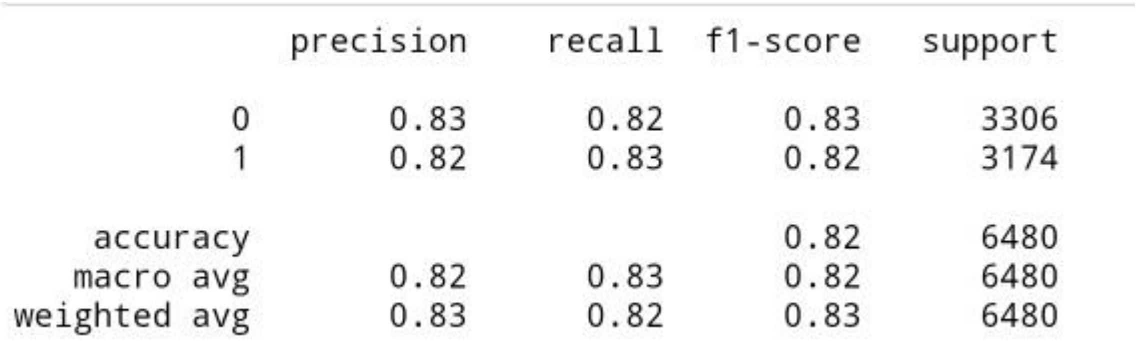
1. RECURRENT NEURAL NETWORK (RNN)

- **RECURRENT NEURAL NETWORK (RNN) with single layer**



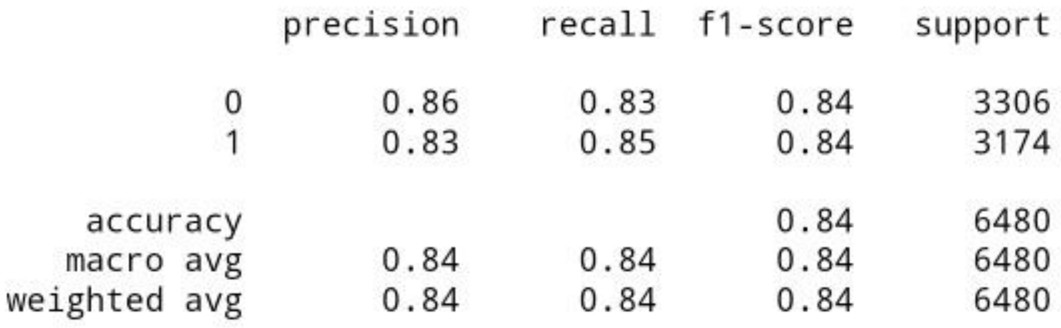
```
              precision    recall  f1-score   support

           0       0.84      0.83      0.83      3306
           1       0.82      0.83      0.83      3174

    accuracy                           0.83      6480
   macro avg       0.83      0.83      0.83      6480
weighted avg       0.83      0.83      0.83      6480
```

- **RECURRENT NEURAL NETWORK (RNN) with Multiple layer**



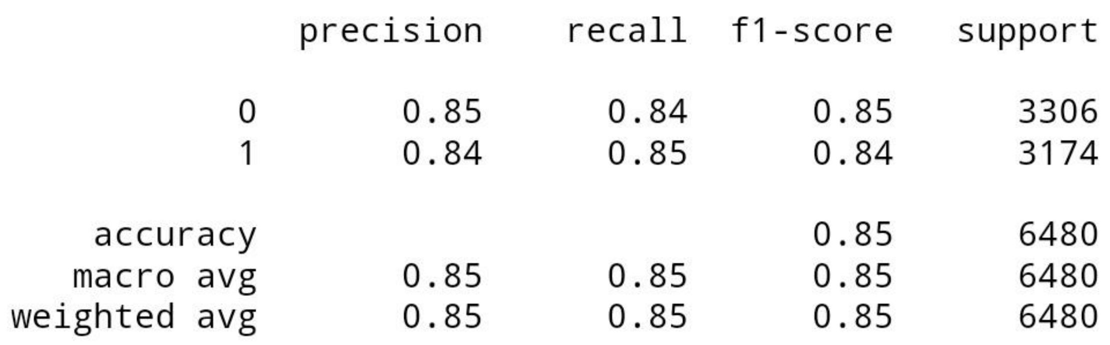
```
              precision    recall  f1-score   support

           0       0.83      0.82      0.83      3306
           1       0.82      0.83      0.82      3174

    accuracy                           0.82      6480
   macro avg       0.82      0.83      0.82      6480
weighted avg       0.83      0.82      0.83      6480
```

- **Bidirectional RECURRENT NEURAL NETWORK (RNN) with single layer**.



```
              precision    recall  f1-score   support

           0       0.86      0.83      0.84      3306
           1       0.83      0.85      0.84      3174

    accuracy                           0.84      6480
   macro avg       0.84      0.84      0.84      6480
weighted avg       0.84      0.84      0.84      6480
```

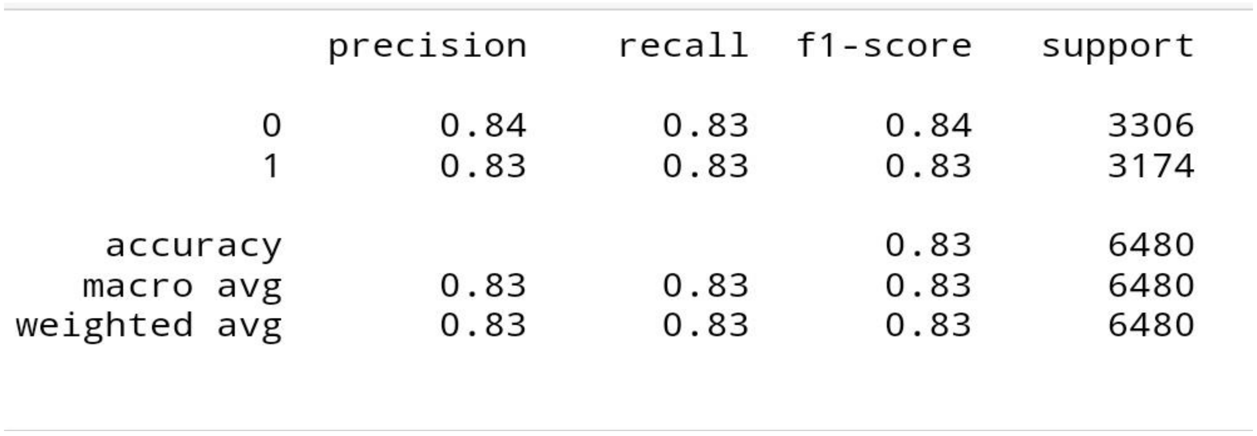
- **Bidirectional RECURRENT NEURAL NETWORK (RNN) with Multiple layer**.



```
              precision    recall  f1-score   support

           0       0.85      0.84      0.85      3306
           1       0.84      0.85      0.84      3174

    accuracy                           0.85      6480
   macro avg       0.85      0.85      0.85      6480
weighted avg       0.85      0.85      0.85      6480
```

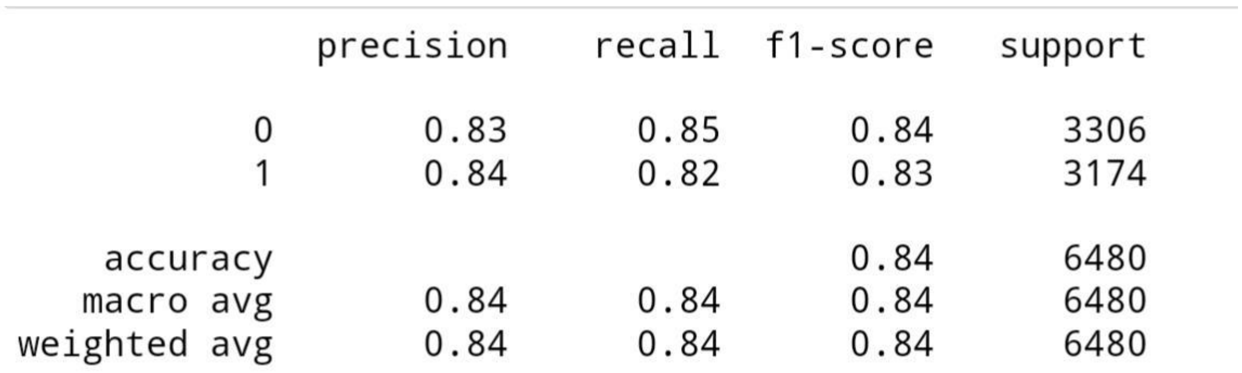
2. LONG SHORT-TERM MEMORY (LSTM)

- **LONG SHORT-TERM MEMORY (LSTM) with single layer**



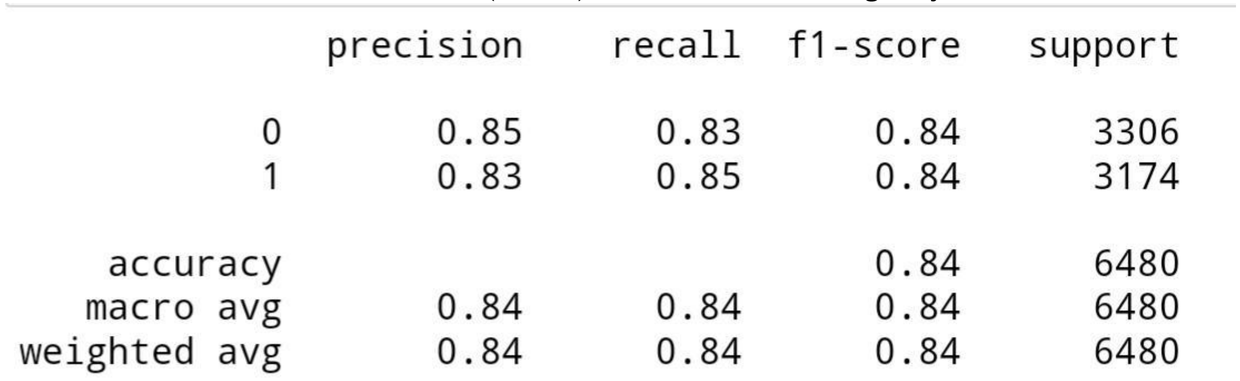
```
              precision    recall  f1-score   support

           0       0.84      0.83      0.84      3306
           1       0.83      0.83      0.83      3174

    accuracy                           0.83      6480
   macro avg       0.83      0.83      0.83      6480
weighted avg       0.83      0.83      0.83      6480
```

- **LONG SHORT-TERM MEMORY (LSTM) with multiple layer**.



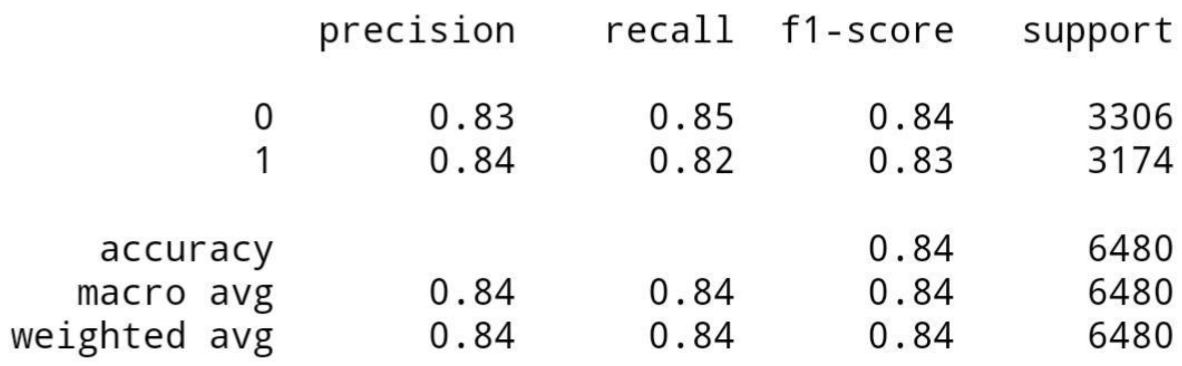
```
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      3306
           1       0.84      0.82      0.83      3174

    accuracy                           0.84      6480
   macro avg       0.84      0.84      0.84      6480
weighted avg       0.84      0.84      0.84      6480
```

- **LONG SHORT-TERM MEMORY (LSTM) Bidirectional with single layer**.



```
              precision    recall  f1-score   support

           0       0.85      0.83      0.84      3306
           1       0.83      0.85      0.84      3174

    accuracy                           0.84      6480
   macro avg       0.84      0.84      0.84      6480
weighted avg       0.84      0.84      0.84      6480
```

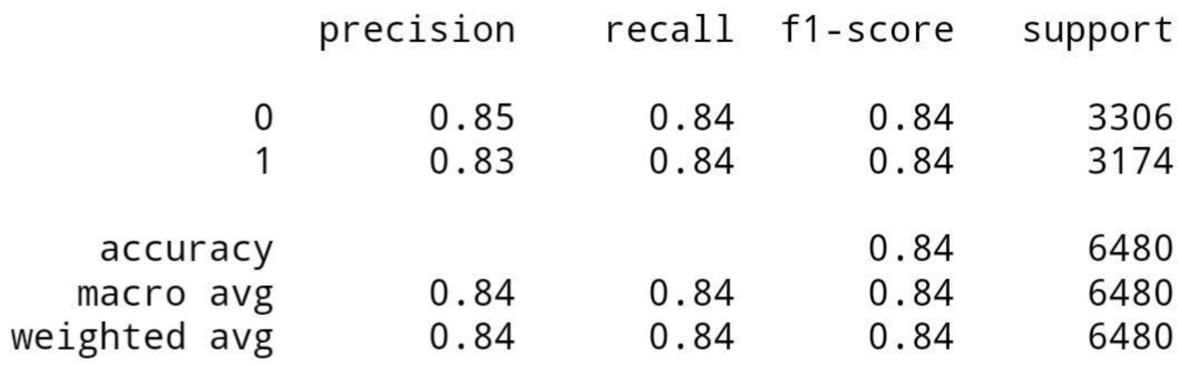
- **LONG SHORT-TERM MEMORY (LSTM) Bidirectional with multiple layer**.



```
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      3306
           1       0.84      0.82      0.83      3174

    accuracy                           0.84      6480
   macro avg       0.84      0.84      0.84      6480
weighted avg       0.84      0.84      0.84      6480
```

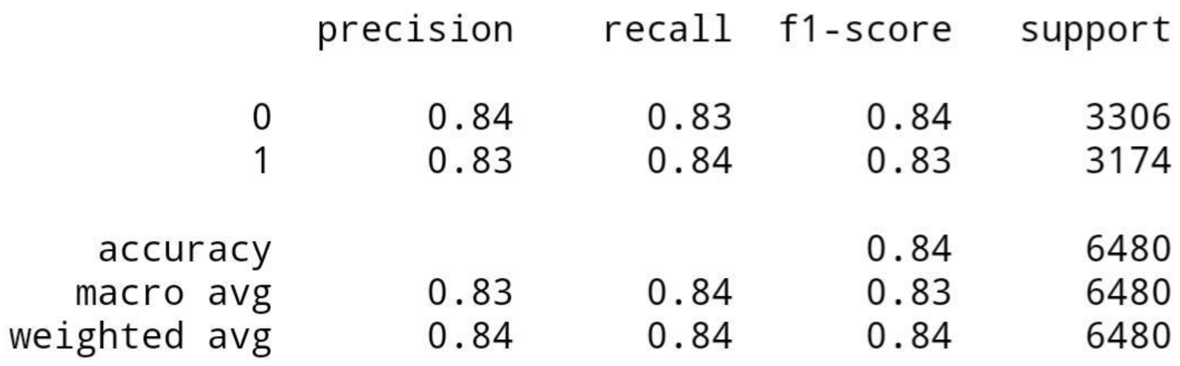
3. GATED RECURRENT UNITS (GRU)

- **GATED RECURRENT UNITS (GRU) with multiple Layer**



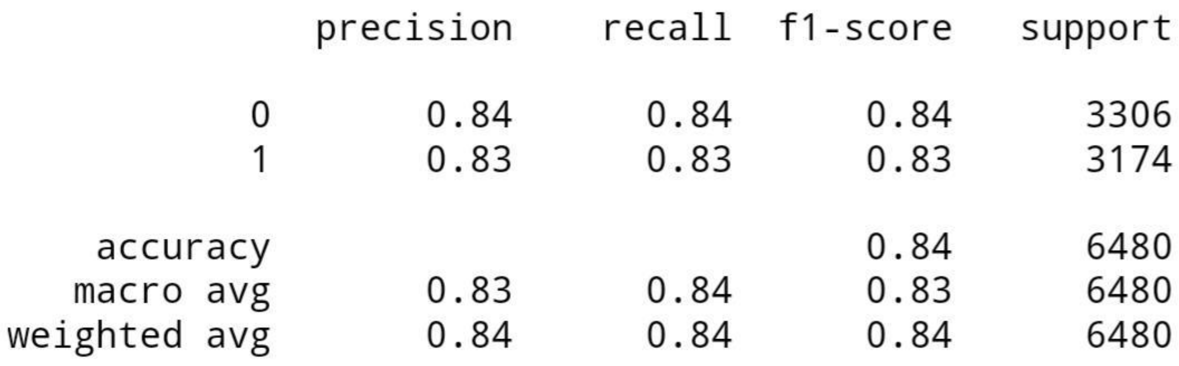
```
              precision    recall  f1-score   support

           0       0.85      0.84      0.84      3306
           1       0.83      0.84      0.84      3174

    accuracy                           0.84      6480
   macro avg       0.84      0.84      0.84      6480
weighted avg       0.84      0.84      0.84      6480
```

- **Bidirectional GATED RECURRENT UNITS (GRU) with single layer**.



```
              precision    recall  f1-score   support

           0       0.84      0.83      0.84      3306
           1       0.83      0.84      0.83      3174

    accuracy                           0.84      6480
   macro avg       0.83      0.84      0.83      6480
weighted avg       0.84      0.84      0.84      6480
```

- **Bidirectional GATED RECURRENT UNITS (GRU) with multiple layer**



```
              precision    recall  f1-score   support

           0       0.84      0.84      0.84      3306
           1       0.83      0.83      0.83      3174

    accuracy                           0.84      6480
   macro avg       0.83      0.84      0.83      6480
weighted avg       0.84      0.84      0.84      6480
```

Conclusion.

Step 1. Remove/drop Unwanted column.

Step 2. Convert Score / Reviews 1 if score is > 3 else 0.

Step 3. Check if there is any website link or any HTML tag are present or not.

Step 4. Perform Preprocessing on Text column remove website links, stop words, etc.

Step 5. Plot image with word cloud to observe what words are repeated many times.

Step 6. Split data into train and test.</b>

Step 7. Perform Tokenization on training data.</b>

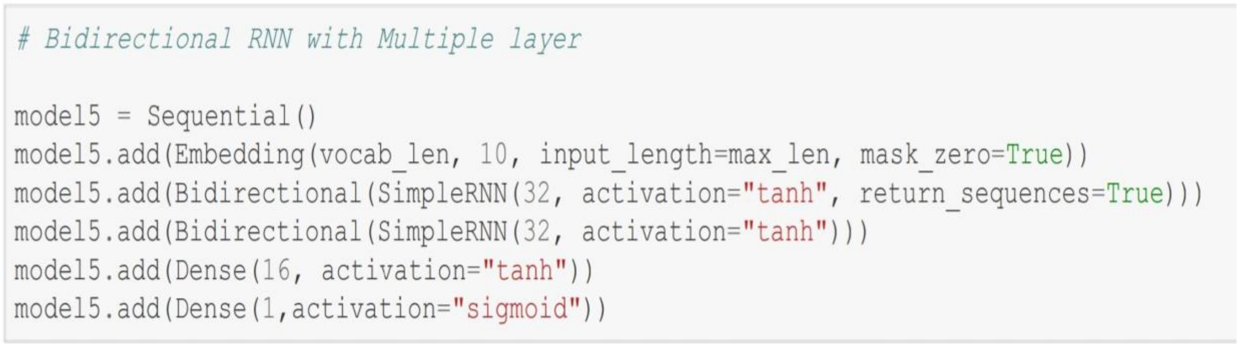
Step 8. Get the maximum length which is covered by 95% of data.

Step 9. Create Neural Network with Flatten and observer the score of testing data.

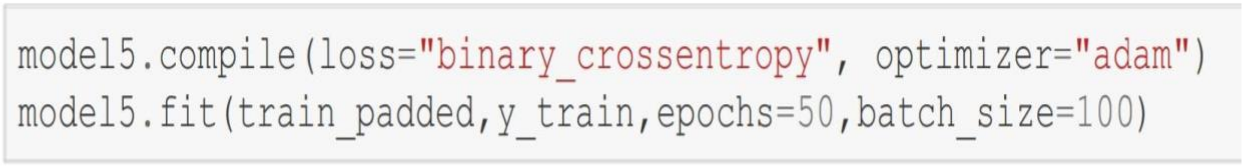
Step 10. After performing NN Perform RECURRENT NEURAL NETWORK (RNN), LONG SHORT-TERM MEMORY (LSTM), and GATED RECURRENT UNITS (GRU) and observe the score & try to improve the recall/score.

ALL OVER RESULT

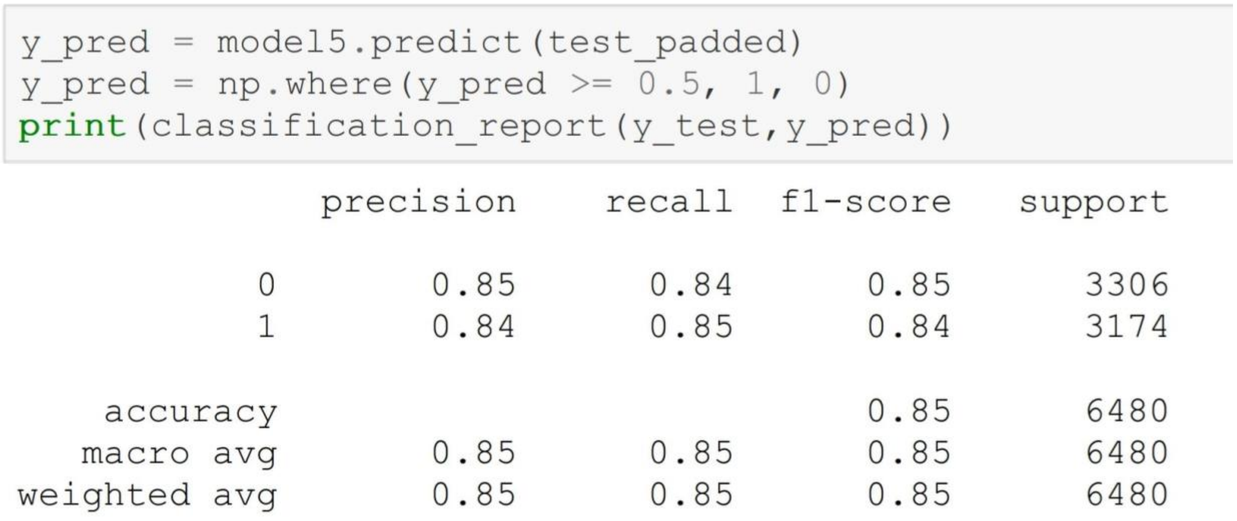
In this Project I have perform almost all the model: With Neural Network we get accuracy score of 81% we perform RECURRENT NEURAL NETWORK (RNN), LONG SHORT-TERM MEMORY (LSTM), GATED RECURRENT UNITS (GRU) with single, multiple and Bidirectional Layer as well to improve the score & we get the **Best score** with **Bidirectional RECURRENT NEURAL NETWORK (RNN) with Multiple layer** of **85% of recall as well as 85% of accuracy.**



```
# Bidirectional RNN with Multiple layer

model5 = Sequential()
model5.add(Embedding(vocab_len, 10, input_length=max_len, mask_zero=True))
model5.add(Bidirectional(SimpleRNN(32, activation="tanh", return_sequences=True)))
model5.add(Bidirectional(SimpleRNN(32, activation="tanh")))
model5.add(Dense(16, activation="tanh"))
model5.add(Dense(1,activation="sigmoid"))
```



```
model5.compile(loss="binary_crossentropy", optimizer="adam")
model5.fit(train_padded,y_train,epochs=50,batch_size=100)
```



```
y_pred = model5.predict(test_padded)
y_pred = np.where(y_pred >= 0.5, 1, 0)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.84      0.85      3306
           1       0.84      0.85      0.84      3174

    accuracy                           0.85      6480
   macro avg       0.85      0.85      0.85      6480
weighted avg       0.85      0.85      0.85      6480
```