```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")

import re
from bs4 import BeautifulSoup
from tqdm import tqdm
from nltk.stem import WordNetLemmatizer

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,f1_score, confusion_matrix

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense , Input , LSTM , Embedding, Dropout , Activation, GRU, Flatten
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.models import Model, Sequential
from keras.layers import Convolution1D
from keras import initializers, regularizers, constraints, optimizers, layers

import nltk
nltk.download('wordnet')
```

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

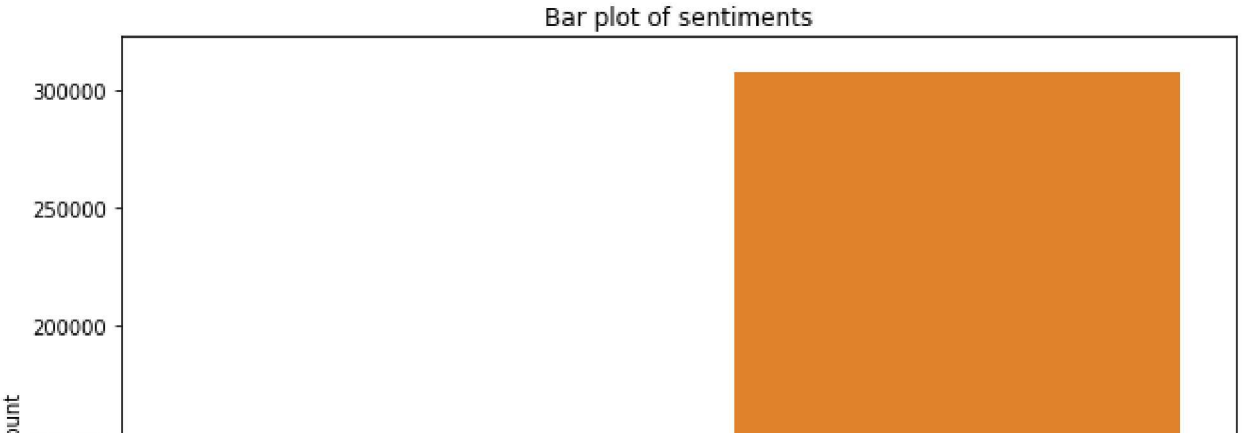
```
df = pd.read_csv('/content/drive/My Drive/Reviews.csv')
df.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	

Natalia

```
# # Removing neutral reviews i.e reviews with score = 3
df_filtered = df[df["Score"]!=3]

# # Defining sentiments based on score - score <= 2 : negative(0), score > 3 : positive(1)
df_filtered["Score"] = df_filtered["Score"].apply(lambda x : 1 if x>3 else 0)

# Removing duplicate reviews
sorted_data=df_filtered.sort_values('ProductId', kind='quicksort', na_position='last')
final_df=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='fir

# Checking Data consistency of HelpfulnessNumerator and HelpfulnessDenominator feature
final_df=final_df[final_df["HelpfulnessNumerator"]<=final_df["HelpfulnessDenominator"]]

# Checking for class imbalance
final_df['Score'].value_counts()

# Plotting based on sentiments
plt.figure(figsize = (10,7))
sns.countplot(final_df['Score'])
plt.title("Bar plot of sentiments")
```

```
Text(0.5, 1.0, 'Bar plot of sentiments')
```



Text Preprocessing

```
# Expanding shortened words to the original form
def decontract(text):
    text = re.sub(r"won\'t", "will not", text)
    text = re.sub(r"can\'t", "can not", text)
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)
    return text
```

```
stop_words= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'y
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll"
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'h
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'unt
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'dur
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'bo
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'ver
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'does
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
            'won', "won't", 'wouldn', "wouldn't"])
```

```
lemmatizer = WordNetLemmatizer()
```

```
def preprocess_text(review):
    review = re.sub(r"http\S+", "", review)             # removing website links
    review = BeautifulSoup(review, 'lxml').get_text()   # removing html tags
    review = decontract(review)                         # decontracting
    review = re.sub("\S*\d\S*", "", review).strip()     # removing the words with numeric dig
    review = re.sub('[^A-Za-z]+', ' ', review)          # removing non-word characters
    review = review.lower()                             # converting to lower case
    review = [word for word in review.split(" ") if not word in stop_words] # removing stop w
    review = [lemmatizer.lemmatize(token, "v") for token in review] #Lemmatization
    review = " ".join(review)
    review.strip()
    return review
final_df['Text'] = final_df['Text'].apply(lambda x: preprocess_text(x))


final_df['Text'].head()

     150523    witty little book make son laugh loud recite c...
     150505    grow read sendak book watch really rosie movie...
     150506    fun way children learn months year learn poems...
     150507    great little book read aloud nice rhythm well ...
     150508    book poetry months year go month cute little p...
     Name: Text, dtype: object


train_df, test_df = train_test_split(final_df, test_size = 0.2, random_state = 42)


train_df.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator
262440	262441	B000NY9QLC	A34449KRZIZR34	M. Reed	5
528673	528674	B000Q6KVIO	A3JN0VZ1JENOB7	BURBERRY GIRL	0

Fitting LSTM with Embedding layer

```
top_words = 6000
tokenizer = Tokenizer(num_words=top_words)
tokenizer.fit_on_texts(train_df['Text'])
list_tokenized_train = tokenizer.texts_to_sequences(train_df['Text'])

max_review_length = 130
X_train = pad_sequences(list_tokenized_train, maxlen=max_review_length)
y_train = train_df['Score']


import tensorflow as tf


y_train
```

```
262440    1
528673    1
566612    0
314532    1
470244    0
         ..
352       1
280604    1
376318    1
567431    1
464546    1
Name: Score, Length: 291336, dtype: int64
```

```
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words+1, embedding_vecor_length, input_length=max_review_length))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
```

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 130, 32)           192032

 lstm (LSTM)                 (None, 100)               53200

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 245,333
Trainable params: 245,333
Non-trainable params: 0
```

```
model.fit(X_train,y_train, epochs=10, batch_size=64, validation_split=0.2)
```

```
Epoch 1/10
3642/3642 [==============================] - 140s 36ms/step - loss: 0.2190 - accuracy: 0
Epoch 2/10
3642/3642 [==============================] - 144s 39ms/step - loss: 0.1696 - accuracy: 0
Epoch 3/10
3642/3642 [==============================] - 136s 37ms/step - loss: 0.1416 - accuracy: 0
Epoch 4/10
3642/3642 [==============================] - 123s 34ms/step - loss: 0.1232 - accuracy: 0
Epoch 5/10
3642/3642 [==============================] - 123s 34ms/step - loss: 0.1091 - accuracy: 0
Epoch 6/10
3642/3642 [==============================] - 127s 35ms/step - loss: 0.0965 - accuracy: 0
Epoch 7/10
3642/3642 [==============================] - 125s 34ms/step - loss: 0.0848 - accuracy: 0
Epoch 8/10
3642/3642 [==============================] - 122s 33ms/step - loss: 0.0740 - accuracy: 0
Epoch 9/10
3642/3642 [==============================] - 120s 33ms/step - loss: 0.0639 - accuracy: 0
Epoch 10/10
3637/3642 [============================>.] - ETA: 0s - loss: 0.0555 - accuracy: 0.9799
```

```
list_tokenized_test = tokenizer.texts_to_sequences(test_df['Text'])
X_test = pad_sequences(list_tokenized_test, maxlen=max_review_length)
y_test = test_df['Score']
prediction = model.predict(X_test)
y_pred = (prediction > 0.5)
print("Accuracy of the model : ", accuracy_score(y_pred, y_test))
print('F1-score: ', f1_score(y_pred, y_test))
print('Confusion matrix:')
confusion_matrix(y_test,y_pred)
```

```
Accuracy of the model :  0.9342074552069747
F1-score:  0.9611933529850021
```

```
Confusion matrix:
array([[ 8697,  2664],
       [ 2128, 59346]])
```

```
a = ["Bad Food"]
a = tokenizer.texts_to_sequences(a)

a = np.array(a)
a = pad_sequences(a, maxlen=max_review_length)

prediction = model.predict(a)
print(prediction)
```

```
[[0.0353176]]
```

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