Sentiment analysis of IMDB reviews We will start by importing the necessary libraries

```
import tensorflow as tf
```

```
import tensorflow.keras as keras
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Importing the data files After importing the necessary libraries now we will read the data files we have two data files here

```
imdb_reviews=pd.read_csv("/content/sample_data/imdb_reviews.csv")
test_reviews=pd.read_csv("/content/sample_data/test_reviews.csv")
```

first data file contains the imdb reviews and their corresponding sentiments which can be either positive or negative, we are going to use this file as our training data.

```
imdb_reviews.head()
```

	Reviews	Sentiment
0	<START this film was just brilliant casting lo...	positive
1	<START big hair big boobs bad music and a gian...	negative
2	<START this has to be one of the worst films o...	negative
3	<START the <UNK> <UNK> at storytelling the tra...	positive
4	<START worst mistake of my life br br i picked...	negative

the second file is also similar to the first file but we are going to use it as the test data.

```
test_reviews.head()
```

	Reviews	Sentiment
0	<START please give this one a miss br br <UNK>...	negative

Preprocessing the data

We can not pass the string data to our model directly, so we need to transform the string data into integer format.For this we can map each distinct word as a distinct integer for eg.{'this':14 , 'the':1}.We already have a file that contains the mapping from words to integers so we are going to load that file.

```
word_index=pd.read_csv("/content/sample_data/word_indexes.csv")
```

The word index file contains mapping from words to integers.

```
word_index.head()
```

	Words	Indexes
0	tsukino	52009
1	nunnery	52010
2	sonja	16819
3	vani	63954
4	woods	1411

Next we are going to convert the word_index dataframe into a python dictionary so that we can use it for converting our reviews from string to integer format.

```
word_index=dict(zip(word_index.Words,word_index.Indexes))
```

```
word_index["<PAD>"]=0
word_index["<START"]=1
word_index["<UNK>"]=2
word_index["<UNUSED>"]=3
```

Now we define a function review_encoder that encodes the reviews into integer format according to the mapping specified by word_index file.

```
def review_encoder(text):
  arr=[word_index[word] for word in text]
  return arr
```

We split the reviews from their corresponding sentiments so that we can preprocess the reviews and sentiments separately and then later pass it to our model.

```
train_data,train_labels=imdb_reviews['Reviews'],imdb_reviews['Sentiment']
test_data, test_labels=test_reviews['Reviews'],test_reviews['Sentiment']
```

Before transforming the reviews as integers we need to tokenize or split the review on the basis of whitespaces For eg.the string "The movie was wonderful" becomes ["The" , "movie" , "was" , "wonderful"].

```
train_data=train_data.apply(lambda review:review.split())
test_data=test_data.apply(lambda review:review.split())
```

Since we have tokenized the reviews now we can apply the review_encoder function to each review and transform the reviews into integer format.

```
train_data=train_data.apply(review_encoder)
test_data=test_data.apply(review_encoder)
```

After transforming, our reviews are going to look like this.

```
train_data.head()
```

```
0    [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, ...
1    [1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463,...
2    [1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5...
3    [1, 4, 2, 2, 33, 2804, 4, 2040, 432, 111, 153,...
4    [1, 249, 1323, 7, 61, 113, 10, 10, 13, 1637, 1...
Name: Reviews, dtype: object
```

We also need to encode the sentiments and we are labeling the positive sentiment as 1 and negative sentiment as 0.

```
def encode_sentiments(x):
  if x=='positive':
    return 1
  else:
    return 0

train_labels=train_labels.apply(encode_sentiments)
test_labels=test_labels.apply(encode_sentiments)
```

Before giving the review as an input to the model we need to perform following preprocessing steps:

The length of each review should be made equal for the model to be working correctly.

We have chosen the length of each review to be 500.

If the review is longer than 500 words we are going to cut the extra part of the review.

If the review is contains less than 500 words we are going to pad the review with zeros to increase its length to 500.

```
train_data=keras.preprocessing.sequence.pad_sequences(train_data,value=word_index["<PAD>"],pa
test_data=keras.preprocessing.sequence.pad_sequences(test_data,value=word_index["<PAD>"],padd
```

Building the model

Our model is a neural network and it consits of the following layers :

one word embedding layer which creates word embeddings of length 16 from integer encoded review.

second layer is global average pooling layer which is used to prevent overfitting by reducing the number of parameters.

then a dense layer which has 16 hidden units and uses relu as activation function

the final layer is the output layer which uses sigmoid as activation function

```
model=keras.Sequential([keras.layers.Embedding(10000,16,input_length=500),
                        keras.layers.GlobalAveragePooling1D(),
                        keras.layers.Dense(16,activation='relu'),
                        keras.layers.Dense(1,activation='sigmoid')])
```

compiling the model

Adam is used as optimization function for our model.

Binary cross entropy loss function is used as loss function for the model.

Accuracy is used as the metric for evaluating the model.

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

In the next step we are going to train the model on our downloaded IMDB dataset.

```
#training the model
```

```
history=model.fit(train_data,train_labels,epochs=30,batch_size=512,validation_data=(test_data
```

```
49/49 [==============================] - 3s 40ms/step - loss: 0.6920 - accuracy: 0.53
Epoch 2/30
49/49 [==============================] - 2s 36ms/step - loss: 0.6853 - accuracy: 0.67
Epoch 3/30
49/49 [==============================] - 2s 36ms/step - loss: 0.6653 - accuracy: 0.72
Epoch 4/30
49/49 [==============================] - 2s 35ms/step - loss: 0.6259 - accuracy: 0.76
Epoch 5/30
49/49 [==============================] - 2s 37ms/step - loss: 0.5705 - accuracy: 0.80
Epoch 6/30
49/49 [==============================] - 2s 36ms/step - loss: 0.5097 - accuracy: 0.83
Epoch 7/30
49/49 [==============================] - 2s 37ms/step - loss: 0.4529 - accuracy: 0.85
Epoch 8/30
49/49 [==============================] - 2s 37ms/step - loss: 0.4055 - accuracy: 0.86
Epoch 9/30
49/49 [==============================] - 2s 36ms/step - loss: 0.3676 - accuracy: 0.87
Epoch 10/30
49/49 [==============================] - 2s 36ms/step - loss: 0.3374 - accuracy: 0.88
Epoch 11/30
49/49 [==============================] - 2s 35ms/step - loss: 0.3146 - accuracy: 0.89
Epoch 12/30
49/49 [==============================] - 2s 36ms/step - loss: 0.2941 - accuracy: 0.89
Epoch 13/30
49/49 [==============================] - 2s 34ms/step - loss: 0.2782 - accuracy: 0.90
Epoch 14/30
49/49 [==============================] - 2s 35ms/step - loss: 0.2634 - accuracy: 0.90
Epoch 15/30
49/49 [==============================] - 2s 34ms/step - loss: 0.2510 - accuracy: 0.91
Epoch 16/30
49/49 [==============================] - 2s 36ms/step - loss: 0.2399 - accuracy: 0.91
Epoch 17/30
49/49 [==============================] - 2s 35ms/step - loss: 0.2298 - accuracy: 0.91
Epoch 18/30
49/49 [==============================] - 2s 36ms/step - loss: 0.2207 - accuracy: 0.92
Epoch 19/30
49/49 [==============================] - 2s 36ms/step - loss: 0.2132 - accuracy: 0.92
Epoch 20/30
49/49 [==============================] - 2s 36ms/step - loss: 0.2059 - accuracy: 0.92
Epoch 21/30
49/49 [==============================] - 2s 35ms/step - loss: 0.1982 - accuracy: 0.93
Epoch 22/30
49/49 [==============================] - 2s 35ms/step - loss: 0.1911 - accuracy: 0.93
Epoch 23/30
49/49 [==============================] - 2s 35ms/step - loss: 0.1850 - accuracy: 0.93
Epoch 24/30
49/49 [==============================] - 2s 35ms/step - loss: 0.1799 - accuracy: 0.93
Epoch 25/30
49/49 [==============================] - 2s 34ms/step - loss: 0.1739 - accuracy: 0.94
Epoch 26/30
49/49 [==============================] - 2s 34ms/step - loss: 0.1688 - accuracy: 0.94
Epoch 27/30
49/49 [==============================] - 2s 35ms/step - loss: 0.1643 - accuracy: 0.94
Epoch 28/30
49/49 [==============================] - 2s 35ms/step - loss: 0.1592 - accuracy: 0.94
```

```
Epoch 29/30
49/49 [==============================] - 2s 34ms/step - loss: 0.1551 - accuracy: 0.94
Epoch 30/30
```

Now we will be evaluating the loss and accuracy of our model on testing data.

```
loss,accuracy=model.evaluate(test_data,test_labels)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.2908 - accuracy: 0.8856
```

As we can see our model is giving an accuracy of 88.58% on the testing data.

Now we are going to take a random review from our test dataset and check wether our model produces correct output or not

```
index=np.random.randint(1,1000)
user_review=test_reviews.loc[index]
print(user_review)
```

```
Reviews      <START my poor tank girl they ignored everythi...
Sentiment                                             negative
Name: 220, dtype: object
```

As we can see the sentiment for the above review is positive, now we are going to take the integer format of this particular review which we already have in our preprocessed test data and then give it as an input to our model to check the prediction of our model.

```
user_review=test_data[index]
user_review=np.array([user_review])
if (model.predict(user_review)>0.5).astype("int32"):
  print("positive sentiment")
else:
  print("negative sentiment")
```

```
negative sentiment
```

As we can see our model is now able to predict the sentiment of the review.

✓ 0s completed at 5:50 PM