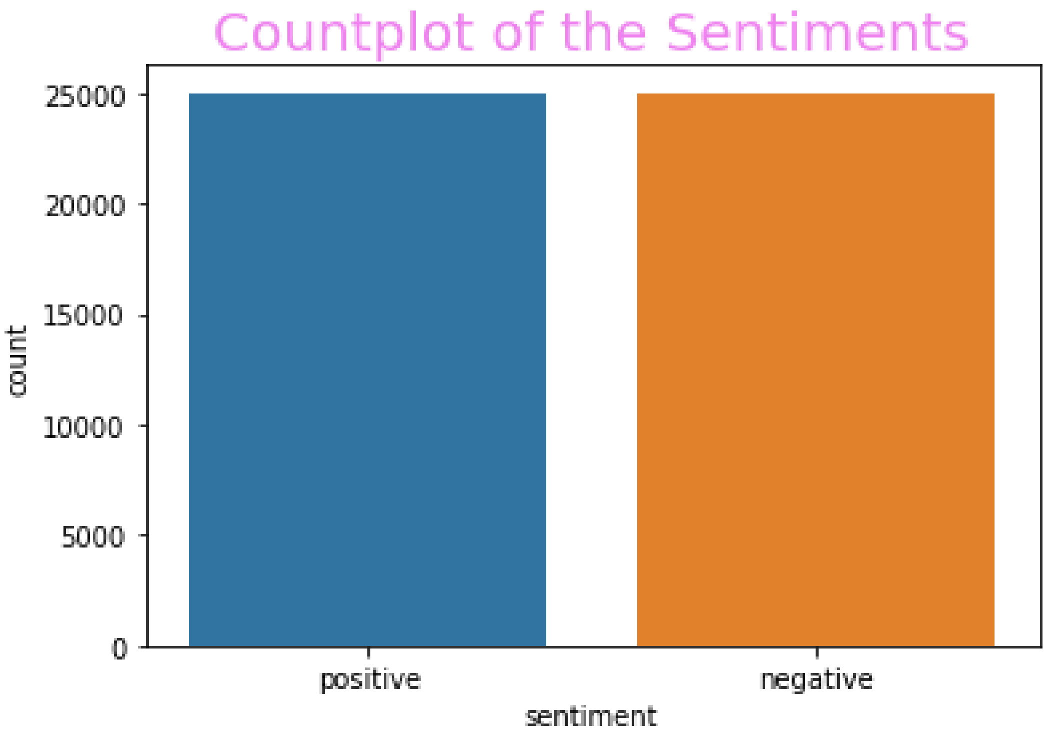
In [13]:
```
#sentiment count
movie['sentiment'].value_counts()
```

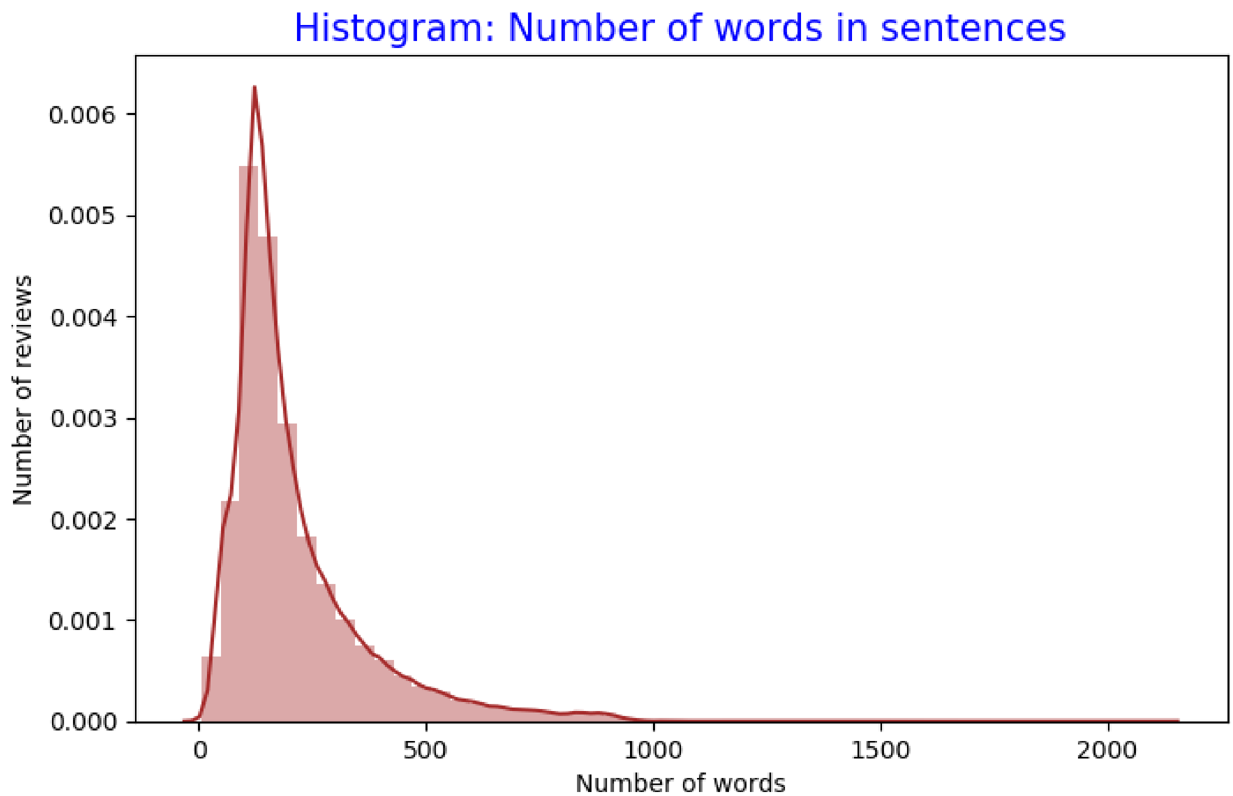
Out[13]:
```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

We can see that the dataset is balanced because it contain equal number of positive and negative reviews.

In [14]:
```
# plotting the labels using seaborn

sns.countplot(movie.sentiment)
plt.title('Countplot of the Sentiments', fontsize = 20, color = 'Violet')
plt.show()
```



In [59]:
```
# Using seaborn to plot the sentence length
# specifying the figure size
# Labeling the title, x axis and y axis
# Displaying the chart

plt.figure(figsize = (8,5), dpi = 100)
sns.distplot(lengths, color = 'brown')
plt.title('Histogram: Number of words in sentences', fontsize = 15, color = 'Blue')
plt.ylabel('Number of reviews')
plt.xlabel('Number of words')
plt.show()
```



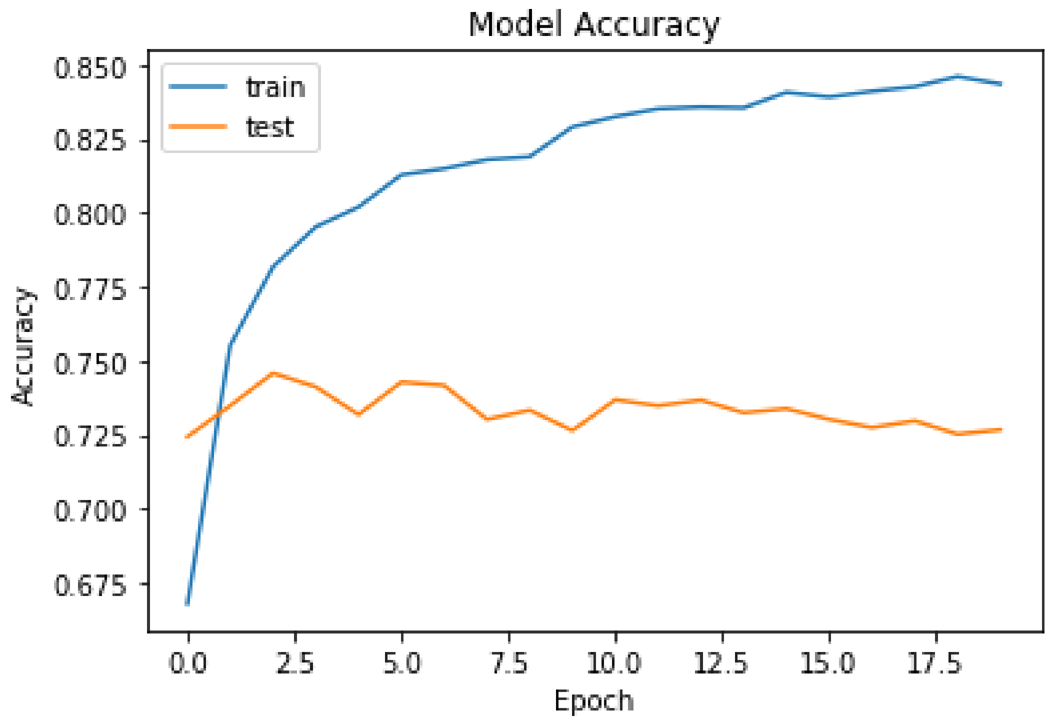
In [69]:
```
# Creating a simple deep neural network
# create an embedding layer by specifying the parameters we created earlier
# Add it to the model
# flatten the embedding layer since we are directly connecting it to densely connected layer
# At the end we add a dense layer with sigmoid activation function.

model = Sequential()

embedding_layer = Embedding(vocab_size, 100, weights = [embedding_matrix],
                            input_length = maxlength, trainable = False)

model.add(embedding_layer)

model.add(Flatten())

model.add(Dense(1, activation = 'sigmoid'))
```

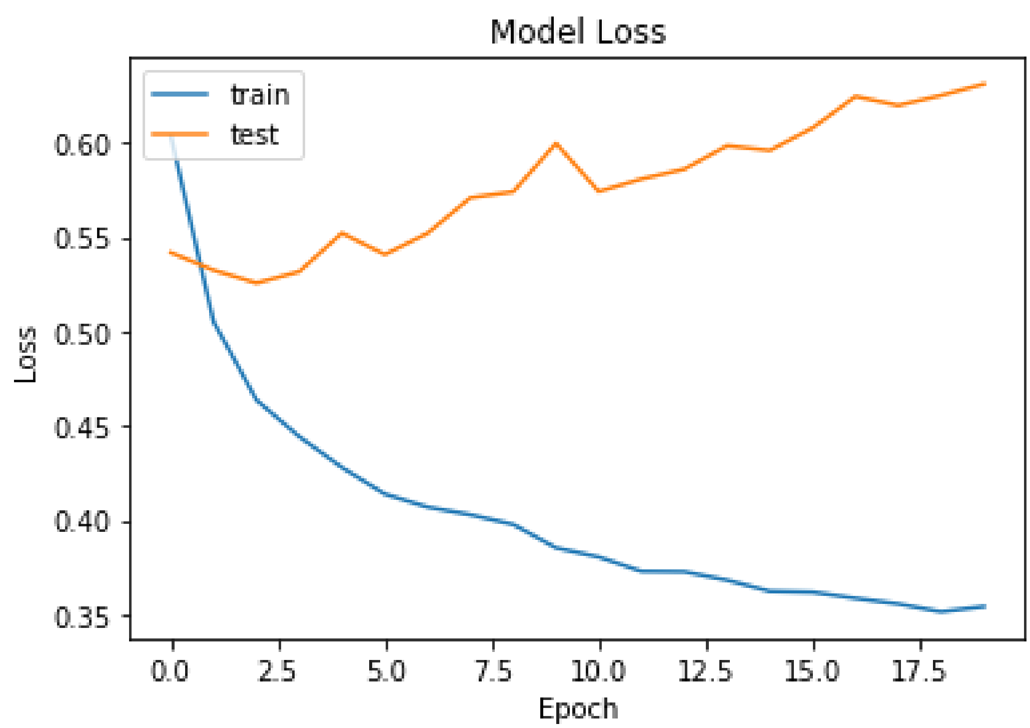
In [70]:
```
# Compiling our model
# Compile defines the loss function, the optimizer and the metrics.

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

# printing the summary of information about our model
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 100)          9254700
_________________________________________________________________
flatten_1 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 10001
=================================================================
Total params: 9,264,701
Trainable params: 10,001
Non-trainable params: 9,254,700
_________________________________________________________________
None
```

In [77]:

```
# Plotting the loss and accuracy differences for training and test sets
# Using matplotlib plotting library
# Giving a title to our chart
# Labeling the x and y axis
# creating a legend
# displaying the chart

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])

plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()
```


Model Accuracy
train
test
0.850
0.825
0.800
0.775
0.750
0.725
0.700
0.675
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


Model Loss
train
test
0.60
0.55
0.50
0.45
0.40
0.35
Loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

In [78]:

```
# Building the model architecture
# Create an embedding layer by specifying the parameters we created earlier
# Creating a Recurrent neural network
# Here we will use LTSM (Long Term Short Term Memory)
# Bidirectional means the RNN processes sequence from start to end,and also backwards
# This makes the model perform better.
# We added another hidden layer and included an activation function as relu.s
# At the end we add a dense layer with sigmoid activation function.
#

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, 100,
                              weights=[embedding_matrix],
                              input_length=maxlength, trainable=False),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

In [79]:

```
# Compiling the model
# Here we specify the loss, optimizer and metrics appropriately.

model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

In [80]:

```
# printing the model summary to view paramaters
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 100)          9254700
_________________________________________________________________
bidirectional (Bidirectional (None, 256)               234496
_________________________________________________________________
dense (Dense)                (None, 128)               32896
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 9,522,221
Trainable params: 267,521
Non-trainable params: 9,254,700
```

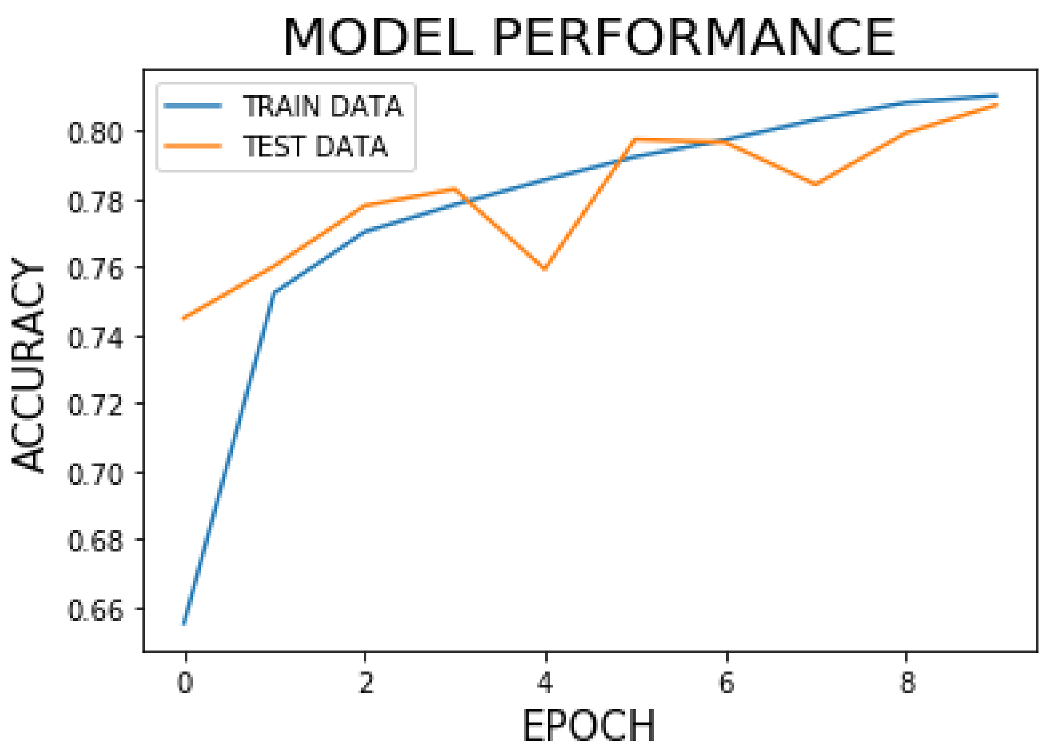
In [84]:

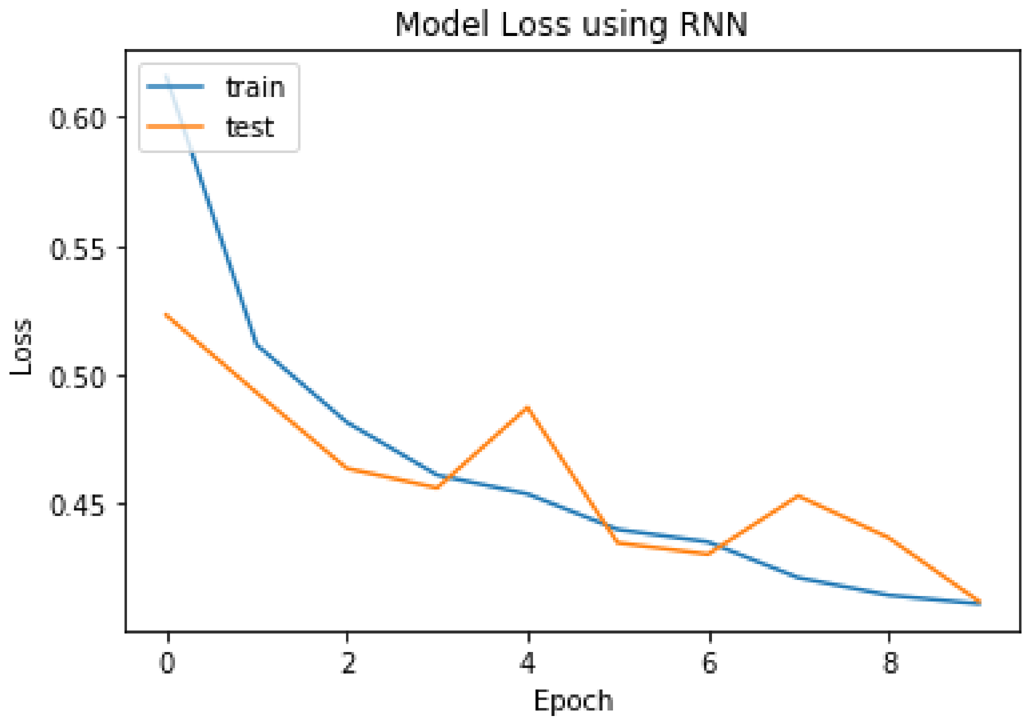
```
# Making predictions

y_pred = model.predict_classes(X_test)

y_pred

# printing the classification report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.84      0.81      4961
           1       0.83      0.78      0.80      5039

    accuracy                           0.81     10000
   macro avg       0.81      0.81      0.81     10000
weighted avg       0.81      0.81      0.81     10000
```

In [88]:

```
# Plotting the loss and accuracy differences for training and test sets
# Using matplotlib plotting library
# Giving a title to our chart
# Labeling the x and y axis
# creating a legend
# displaying the chart


plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])

plt.title('MODEL PERFORMANCE', fontsize = 20)
plt.ylabel('ACCURACY', fontsize = 15)
plt.xlabel('EPOCH', fontsize = 15)
plt.legend(['TRAIN DATA', 'TEST DATA'], loc = 'upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('Model Loss using RNN')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()
```





In [89]:
```
# Building the model
# Create an embedding layer by specifying the parameters we created earlier
# Creating two Recurrent neural network layers
# Here we will use two LTSM (Long Term Short Term Memory) layers
# Bidirectional means the RNN processes sequence from start to end,and also backwards
# This makes the model perform better.
# We added another hidden layer and included an activation function as relu.
# Here we added Dropout to prevent the model from over fitting
# Dropout randomly removes some neurons in the hidden layers
# At the end we add a dense layer with sigmoid activation function.

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, 100,
                              weights=[embedding_matrix],
                              input_length=maxlength, trainable=False),

    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(50, return_sequences = True)),

    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(25)),

    tf.keras.layers.Dense(50, activation='relu'),

    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(1, activation='sigmoid')])
```

In [90]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 100)          9254700
_________________________________________________________________
bidirectional_1 (Bidirection (None, 100, 100)          60400
_________________________________________________________________
bidirectional_2 (Bidirection (None, 50)                25200
_________________________________________________________________
dense_2 (Dense)              (None, 50)                2550
_________________________________________________________________
dropout (Dropout)            (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 51
=================================================================
Total params: 9,342,901
Trainable params: 88,201
Non-trainable params: 9,254,700
_________________________________________________________________
```

In [103]:
```
model = Sequential()

embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlength , trainable=False)
model.add(embedding_layer)

model.add(Conv1D(128, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dropout(0.2)),
model.add(Dense(1, activation='sigmoid'))
```

In [104]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

In [105]:
```
print(model.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 100, 100)          9254700
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 96, 128)           64128
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 9,318,957
Trainable params: 64,257
Non-trainable params: 9,254,700
_________________________________________________________________
None
```

Conclusion

. Accuracy score on CNN is 83.21%

. Accuracy score using Simple Deep Neural Networks is 72.35%

. Accuracy using Recurrent Neural Network (RNN) with LSTM and Dropout is 80%

. Accuracy using 2 RNN layer with LSTM is 82%

. CNN model gives nest accuracy among all with 83.21% on training and testing dataset.

. Using the RNN (LSTM) with two hidden layers is also best which yield 82% accuracy on both training and testing dataset.

. Neural Network models effective for sentiment analysis on IMDB reviews.

Optimizing the model further may yield better result using more data.