Source Code and Output-

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
#loading imdb data with most frequent 10000 words
from keras.datasets import imdb
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000) # you may take top 10,000 word frequently used review of movies other are discarded
#consolidating data for EDA Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics
data = np.concatenate((X_train, X_test), axis=0) # axis 0 is first running vertically downwards across rows (axis 0), axis 1 is second running horizontally across columns (axis 1), label = np.concatenate((y_train, y_test), axis=0)
X_train.shape
(25000,)
X_test.shape
(25000,)
y_train.shape
(25000,)
y_test.shape
(25000,)
print("Review is ",X_train[0]) # series of no converted word to vocabulory associated with index
print("Review is ",y_train[0])
Review is [1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11, 3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123, 125, 68, 2, 6853, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 8255, 5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64, 1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 95]
Review is  0
vocab=imdb.get_word_index() # Retrieve the word index file mapping words to indices print(vocab)
{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951, 'woods': 1408, 'spiders': 16115,
y_train
```

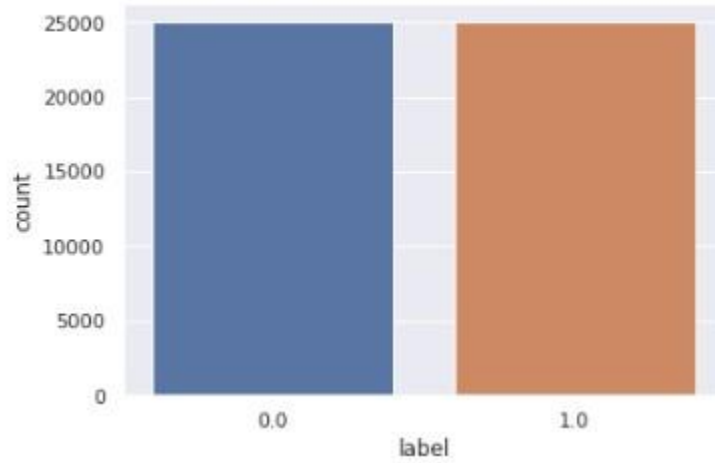
```
array([1, 0, 0, ..., 0, 1, 0])
y_test
array([0, 1, 1, ..., 0, 0, 0])
def vectorize(sequences, dimension = 10000): # We will vectorize every review and fill it with zeros so that it contains exactly 10,000 numbers.
  # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension)) for i, sequence
    in enumerate(sequences):
        results[i, sequence] = 1
    return results
test_x = data[:10000]
test_y = label[:10000]
train_x = data[10000:]
train_y = label[10000:]
test_x.shape (10000,)
test_y.shape (10000,)
train_x.shape (40000,)
train_y.shape (40000,)
print("Categories:", np.unique(label))
print("Number of unique words:", len(np.unique(np.hstack(data))))
# The hstack() function is used to stack arrays in sequence horizontally (column wise).
Categories: [0 1]
Number of unique words: 9998
length = [len(i) for i in data]
print("Average Review length:", np.mean(length))
print("Standard Deviation:", round(np.std(length)))

# The whole dataset contains 9998 unique words and the average review length is 234 words, with a standard deviation of 173 words.
Average Review length: 234.75892
Standard Deviation: 173
print("Label:", label[0])
Label: 1
print("Label:", label[1])
Label: 0 print(data[0])
# Retrieves a dict mapping words to their index in the IMDB dataset.
```

```
index = imdb.get_word_index() # word to index
reverse_index = dict([(value, key) for (key, value) in index.items()]) # id to word

decoded = " ".join( [reverse_index.get(i - 3, "#") for i in data[0]] )
data = vectorize(data)
label = np.array(label).astype("float32")
labelDF=pd.DataFrame({'label':label})
sns.countplot(x='label', data=labelDF)
```

```
<AxesSubplot:xlabel='label', ylabel='count'>
```




```
# Creating train and test data set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,label, test_size=0.20, random_state=1)
X_train.shape
(40000, 10000)
X_test.shape
(10000, 10000)
# Let's create sequential model from
keras.utils import to_categorical from
keras import models from keras
import layers model =
models.Sequential()
# Input - Layer
# Note that we set the input-shape to 10,000 at the input-layer because our reviews are 10,000 integers long.
# The input-layer takes 10,000 as input and outputs it with a shape of 50.
model.add(layers.Dense(50, activation = "relu", input_shape=(10000, ))) # Hidden - Layers
```

```
# Please note you should always use a dropout rate between 20% and 50%. # here in our case 0.3 means 30% dropout we are using dropout to prevent overfitting.
# By the way, if you want you can build a sentiment analysis without LSTMs, then you simply need to replace it by a flatten layer:
model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
# Output- Layer
model.add(layers.Dense(1, activation = "sigmoid"))
import tensorflow as tf callback =
tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
model.compile(
 optimizer = "adam", loss =
 "binary_crossentropy",
 metrics = ["accuracy"]
) from sklearn.model_selection import
train_test_split

results = model.fit(
 X_train, y_train, epochs= 2,
 batch_size = 500, validation_data
 = (X_test, y_test),
 callbacks=[callback]
) # Let's check mean accuracy of our model
print(np.mean(results.history["val_accuracy"])) #
Evaluate the model score = model.evaluate(X_test,
y_test, batch_size=500) print('Test loss:', score[0])
print('Test accuracy:', score[1])
20/20 [==============================] - 1s 24ms/step - loss: 0.2511 - accuracy:
0.8986
Test loss: 0.25108325481414795
Test accuracy: 0.8985999822616577 #Let's
plot training history of our model.

# list all data in history
print(results.history.keys()) # summarize
history for accuracy
plt.plot(results.history['accuracy'])
plt.plot(results.history['val_accuracy'])
```

```
plt.title('model accuracy')
plt.ylabel('accuracy') plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show() # summarize history for loss
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss'])
plt.title('model loss') plt.ylabel('loss')
plt.xlabel('epoch') plt.legend(['train',
'test'], loc='upper left') plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```