In [1]:
```
# The IMDB sentiment classification dataset consists of 50,000 movie reviews f
# The reviews are preprocessed and each one is encoded as a sequence of word i
# The words within the reviews are indexed by their overall frequency within t
# The 50,000 reviews are split into 25,000 for training and 25,000 for testing
# Text Process word by word at diffrent timestamp ( You may use RNN(Recurrent
# convert input text to vector reprent input text
# DOMAIN: Digital content and entertainment industry
# CONTEXT: The objective of this project is to build a text classification mod
# DATA DESCRIPTION: The Dataset of 50,000 movie reviews from IMDB, labelled by
# Reviews have been preprocessed, and each review is encoded as a sequence of
# For convenience, the words are indexed by their frequency in the dataset, me
# Use the first 20 words from each review to speed up training, using a max vo
# As a convention, "0" does not stand for a specific word, but instead is used

# PROJECT OBJECTIVE: Build a sequential NLP classifier which can use input tex
import numpy as np
import pandas as pd
```

In [2]:
```
#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 m
#consolidating data for EDA(exploratory data analysis: involves gathering all
data = np.concatenate((X_train, X_test), axis=0)
label = np.concatenate((y_train, y_test), axis=0)
```

```
WARNING:tensorflow:From C:\Users\Sakshi's PC\AppData\Roaming\Python\Python311
\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cr
oss_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cros
s_entropy instead.

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-data
sets/imdb.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/im
db.npz)
17464789/17464789 [==============================] - 8s 0us/step
```

In [3]:
```
print("Review is ",X_train[5]) # series of no converted word to vocabulory ass
print("Review is ",y_train[5]) # 0 indicating a negative review and 1 indicati
```

```
Review is  [1, 778, 128, 74, 12, 630, 163, 15, 4, 1766, 7982, 1051, 2, 32, 8
5, 156, 45, 40, 148, 139, 121, 664, 665, 10, 10, 1361, 173, 4, 749, 2, 16, 38
04, 8, 4, 226, 65, 12, 43, 127, 24, 2, 10, 10]
Review is  0
```

In [4]:
```
vocab=imdb.get_word_index() #The code you provided retrieves the word index fo
print(vocab)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-d
atasets/imdb_word_index.json (https://storage.googleapis.com/tensorflow/tf
-keras-datasets/imdb_word_index.json)
1641221/1641221 [==============================] - 1s 0us/step
{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'van
i': 63951, 'woods': 1408, 'spiders': 16115, 'hanging': 2345, 'woody': 228
9, 'trawling': 52008, "hold's": 52009, 'comically': 11307, 'localized': 40
830, 'disobeying': 30568, "'royale": 52010, "harpo's": 40831, 'canet': 520
11, 'aileen': 19313, 'acurately': 52012, "diplomat's": 52013, 'rickman': 2
5242, 'arranged': 6746, 'rumbustious': 52014, 'familiarness': 52015, "spid
er'": 52016, 'hahahah': 68804, "wood'": 52017, 'transvestism': 40833, "han
gin'": 34702, 'bringing': 2338, 'seamier': 40834, 'wooded': 34703, 'bravor
a': 52018, 'grueling': 16817, 'wooden': 1636, 'wednesday': 16818, "'prix":
52019, 'altagracia': 34704, 'circuitry': 52020, 'crotch': 11585, 'busybod
y': 57766, "tart'n'tangy": 52021, 'burgade': 14129, 'thrace': 52023, "to
m's": 11038, 'snuggles': 52025, 'francesco': 29114, 'complainers': 52027,
'templarios': 52125, '272': 40835, '273': 52028, 'zaniacs': 52130, '275':
34706, 'consenting': 27631, 'snuggled': 40836, 'inanimate': 15492, 'ualit
y': 52030, 'bronte': 11926, 'errors': 4010, 'dialogs': 3230, "yomada's": 5
```

In [5]:
```
data #data is a numpy array that contains all the text data from the IMDB data
```

Out[5]:
```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 394
1, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5,
150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13,
447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71,
87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 1
8, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 3
3, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22,
12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4,
2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 1
3, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4,
226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 2
5, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32,
15, 16, 5345, 19, 178, 32]),
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 2
6, 4, 715, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 31
03, 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, 1
649, 26, 6853, 5, 163, 11, 3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 26
37, 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, 169
0, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 95]),
       list([1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5974, 54, 61, 369, 1
3, 71, 149, 14, 22, 112, 4, 2401, 311, 12, 16, 3711, 33, 75, 43, 1829, 296,
4, 86, 320, 35, 534, 19, 263, 4821, 1301, 4, 1873, 33, 89, 78, 12, 66, 16, 4,
360, 7, 4, 58, 316, 334, 11, 4, 1716, 43, 645, 662, 8, 257, 85, 1200, 42, 122
8, 2578, 83, 68, 3912, 15, 36, 165, 1539, 278, 36, 69, 2, 780, 8, 106, 14, 69
05, 1338, 18, 6, 22, 12, 215, 28, 610, 40, 6, 87, 326, 23, 2300, 21, 23, 22,
12, 272, 40, 57, 31, 11, 4, 22, 47, 6, 2307, 51, 9, 170, 23, 595, 116, 595, 1
352, 13, 191, 79, 638, 89, 2, 14, 9, 8, 106, 607, 624, 35, 534, 6, 227, 7, 12
9, 113]),
       ...,
       list([1, 13, 1408, 15, 8, 135, 14, 9, 35, 32, 46, 394, 20, 62, 30, 509
3, 21, 45, 184, 78, 4, 1492, 910, 769, 2290, 2515, 395, 4257, 5, 1454, 11, 11
9, 2, 89, 1036, 4, 116, 218, 78, 21, 407, 100, 30, 128, 262, 15, 7, 185, 228
0, 284, 1842, 2, 37, 315, 4, 226, 20, 272, 2942, 40, 29, 152, 60, 181, 8, 30,
50, 553, 362, 80, 119, 12, 21, 846, 5518]),
       list([1, 11, 119, 241, 9, 4, 840, 20, 12, 468, 15, 94, 3684, 562, 791,
39, 4, 86, 107, 8, 97, 14, 31, 33, 4, 2960, 7, 743, 46, 1028, 9, 3531, 5, 4,
768, 47, 8, 79, 90, 145, 164, 162, 50, 6, 501, 119, 7, 9, 4, 78, 232, 15, 16,
224, 11, 4, 333, 20, 4, 985, 200, 5, 2, 5, 9, 1861, 8, 79, 357, 4, 20, 47, 22
0, 57, 206, 139, 11, 12, 5, 55, 117, 212, 13, 1276, 92, 124, 51, 45, 1188, 7
1, 536, 13, 520, 14, 20, 6, 2302, 7, 470]),
       list([1, 6, 52, 7465, 430, 22, 9, 220, 2594, 8, 28, 2, 519, 3227, 6, 7
69, 15, 47, 6, 3482, 4067, 8, 114, 5, 33, 222, 31, 55, 184, 704, 5586, 2, 19,
346, 3153, 5, 6, 364, 350, 4, 184, 5586, 9, 133, 1810, 11, 5417, 2, 21, 4, 72
98, 2, 570, 50, 2005, 2643, 9, 6, 1249, 17, 6, 2, 2, 21, 17, 6, 1211, 232, 11
38, 2249, 29, 266, 56, 96, 346, 194, 308, 9, 194, 21, 29, 218, 1078, 19, 4, 7
8, 173, 7, 27, 2, 5698, 3406, 718, 2, 9, 6, 6907, 17, 210, 5, 3281, 5677, 47,
77, 395, 14, 172, 173, 18, 2740, 2931, 4517, 82, 127, 27, 173, 11, 6, 392, 21
7, 21, 50, 9, 57, 65, 12, 2, 53, 40, 35, 390, 7, 11, 4, 3567, 7, 4, 314, 74,
6, 792, 22, 2, 19, 714, 727, 5205, 382, 4, 91, 6533, 439, 19, 14, 20, 9, 144
1, 5805, 1118, 4, 756, 25, 124, 4, 31, 12, 16, 93, 804, 34, 2005, 2643])],
      dtype=object)
```

In [6]:
```
label #label is a numpy array that contains all the sentiment labels from the
```

Out[6]:
```
array([1, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [7]:
```
X_train.shape
```

Out[7]:
```
(25000,)
```

In [8]:
```
X_test.shape
```

Out[8]:
```
(25000,)
```

In [9]:
```
y_train
```

Out[9]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

In [10]:
```
y_test # y_test is 25000, which indicates that it contains 25000 sentiment lab
```

Out[10]:
```
array([0, 1, 1, ..., 0, 0, 0], dtype=int64)
```

In [11]:
```
# Function to perform relevant sequence adding on the data
# Now it is time to prepare our data. We will vectorize every review and fill
# That means we fill every review that is shorter than 500 with zeros.
# We do this because the biggest review is nearly that long and every input fo
# We also transform the targets into floats.
# sequences is name of method the review less than 1000 we perform padding ove

def vectorize(sequences, dimension = 10000):
  # 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

# The script transforms your dataset into a binary vector space model.
# First, if we examine the x_train content we see that each review is represen
# print(train_data[0])  # print the first review
# [1, 14, 22, 16, 43, 530, 973, ..., 5345, 19, 178, 32]
# the size of the entire dictionary, dictionary=10000 in your example.
# We will then associate each element/index of this vector with one word/word_
# So word represented by word id 14 will now be represented by 14-th element o


# Each element will either be 0 (word is not present in the review) or 1 (word
# And we can treat this as a probability, so we even have meaning for values i
# Furthermore, every review will now be represented by this very long (sparse)
# word      word_id
# I      -> 0
# you    -> 1
# he     -> 2
# be     -> 3
# eat    -> 4
# happy  -> 5
# sad    -> 6
# banana -> 7
# a      -> 8

# the sentences would then be processed in a following way.


# I be happy      -> [0,3,5]   -> [1,0,0,1,0,1,0,0,0]
# I eat a banana. -> [0,4,8,7] -> [1,0,0,0,1,0,0,1,1]

# Now I highlighted the word sparse.
# That means, there will have A LOT MORE zeros in comparison with ones.
# We can take advantage of that. Instead of checking every word, whether it is
# we will check a substantially smaller list of only those words that DO appea

# Therefore, we can make things easy for us and create reviews x vocabulary ma
# And then just go through words in each review and flip the indicator to 1.0

# result[review_id][word_id] = 1.0
# So instead of doing 25000 x 10000 = 250 000 000 operations,
# we only did number of words = 5 967 841. That's just ~2.5% of original amoun


# The for loop here is not processing all the matrix. As you can see, it enume
```

```
# t = np.array([1,2,3,4,5,6,7,8,9])
# r = np.zeros((len(t), 10))

#Output

# array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
#   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
#   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
#   [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
#  [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
#  [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
#  [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
#  [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
# [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]]) #

# then we modify elements with the same way you have :

# for i, s in enumerate(t):
#  r[i,s] = 1.
# array([[0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
#   [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
#   [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
#   [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
#   [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
#   [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
#   [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
#   [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
#   [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
  # you can see that the for loop modified only a set of elements (len(t))
  # which has index [i,s] (in this case ; (0, 1), (1, 2), (2, 3), an so on))
```

In [12]:
```
# Now we split our data into a training and a testing set.
# The training set will contain  reviews and the testing set

test_x = data[:10000]
test_y = label[:10000]
train_x = data[10000:]
train_y = label[10000:]
```

In [13]:
```
test_x
```

Out[13]:
```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 394
1, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5,
150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13,
447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71,
87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 1
8, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 3
3, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22,
12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4,
2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 1
3, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4,
226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 2
5, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32,
15, 16, 5345, 19, 178, 32]),
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 2
6, 4, 715, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 31
03, 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, 1
649, 26, 6853, 5, 163, 11, 3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 26
37, 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, 169
0, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 95]),
       list([1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5974, 54, 61, 369, 1
3, 71, 149, 14, 22, 112, 4, 2401, 311, 12, 16, 3711, 33, 75, 43, 1829, 296,
4, 86, 320, 35, 534, 19, 263, 4821, 1301, 4, 1873, 33, 89, 78, 12, 66, 16, 4,
360, 7, 4, 58, 316, 334, 11, 4, 1716, 43, 645, 662, 8, 257, 85, 1200, 42, 122
8, 2578, 83, 68, 3912, 15, 36, 165, 1539, 278, 36, 69, 2, 780, 8, 106, 14, 69
05, 1338, 18, 6, 22, 12, 215, 28, 610, 40, 6, 87, 326, 23, 2300, 21, 23, 22,
12, 272, 40, 57, 31, 11, 4, 22, 47, 6, 2307, 51, 9, 170, 23, 595, 116, 595, 1
352, 13, 191, 79, 638, 89, 2, 14, 9, 8, 106, 607, 624, 35, 534, 6, 227, 7, 12
9, 113]),
       ...,
       list([1, 14, 9, 6, 66, 327, 5, 1047, 20, 15, 4, 436, 223, 70, 358, 45,
44, 107, 2515, 5, 6, 1132, 37, 26, 623, 245, 8, 412, 19, 294, 334, 18, 6, 11
7, 137, 21, 4, 1389, 92, 391, 5, 36, 1090, 5, 140, 8, 169, 4, 223, 23, 68, 20
5, 4, 1132, 9, 773, 5621, 5, 59, 456, 56, 8, 41, 403, 580, 9, 4, 1155, 912, 3
7, 694, 6, 176, 44, 113, 23, 4, 1004, 7, 4, 6567, 2694, 9, 4, 922, 5, 2, 912,
37, 5190, 183, 276, 148, 289, 295, 23, 35, 1154, 5, 12, 166, 18, 6, 654, 5, 2
53, 1061, 58, 50, 26, 57, 318, 302, 7, 4, 6849, 728, 38, 12, 218, 954, 33, 3
2, 45, 4, 118, 662, 1626, 20, 15, 207, 110, 38, 230, 45, 6, 66, 52, 20, 18, 2
166]),
       list([1, 14, 20, 9, 43, 160, 856, 206, 509, 21, 12, 100, 28, 77, 38, 7
6, 128, 54, 4, 1865, 216, 46, 36, 66, 887, 49, 3822, 339, 294, 40, 1798, 2, 3
7, 93, 15, 530, 206, 720, 11, 3567, 17, 36, 847, 56, 4, 890, 39, 4, 4565, 62,
28, 679, 4, 753, 206, 844, 83, 142, 318, 88, 4, 360, 7, 4, 22, 16, 2659, 727,
21, 1753, 128, 74, 25, 62, 535, 39, 14, 552, 7, 20, 95, 385, 4, 477, 136, 4,
123, 180, 4, 31, 75, 69, 77, 1064, 18, 21, 16, 40, 149, 142, 39, 4, 6, 768, 1
1, 4, 2084, 36, 1258, 5261, 164, 1936, 5, 36, 521, 187, 6, 313, 269, 8, 516,
257, 85, 172, 154, 172, 154]),
       list([1, 51, 527, 487, 5, 116, 57, 1613, 51, 25, 191, 97, 6, 52, 20, 1
9, 6, 686, 109, 7660, 12, 16, 224, 11, 2, 19, 532, 807, 10, 10, 38, 14, 554,
271, 23, 6, 1189, 8, 67, 27, 336, 4, 554, 1655, 304, 6, 1707, 5, 4, 1811, 47,
6, 483, 1274, 5, 1442, 1696, 2817, 38, 4, 554, 6141, 11, 6, 2144, 5, 6095, 9
5, 29, 1129, 187, 4247, 11, 4, 5152, 366, 29, 214, 6590, 10, 10, 315, 15, 58,
29, 1860, 6, 2123, 1453, 4, 86, 58, 29, 1860, 12, 125, 11, 4, 2144, 4, 333, 5
```

```
8, 29, 166, 6, 2, 46, 7, 6, 9459, 5, 9866, 4, 2123, 107, 665, 7, 1214, 541,
2, 46, 7, 4, 2, 4539, 1578, 72, 7, 6498, 2, 4, 3942, 9997, 10, 10, 82, 4, 55
4, 1068, 8, 1968, 6, 2, 19, 727, 1901, 10, 10, 3288])],
      dtype=object)
```

In [14]:
```
test_y
```

Out[14]:
```
array([1, 0, 0, ..., 1, 0, 0], dtype=int64)
```

In [15]:
```
train_x
```

Out[15]:
```
array([list([1, 13, 104, 14, 9, 31, 7, 4, 4343, 7, 4, 3776, 3394, 2, 495, 10
3, 141, 87, 2048, 17, 76, 2, 44, 164, 525, 13, 197, 14, 16, 338, 4, 177, 16,
6118, 5253, 2, 2, 2, 21, 61, 1126, 2, 16, 15, 36, 4621, 19, 4, 2, 157, 5, 60
5, 46, 49, 7, 4, 297, 8, 276, 11, 4, 621, 837, 844, 10, 10, 25, 43, 92, 81, 2
282, 5, 95, 947, 19, 4, 297, 806, 21, 15, 9, 43, 355, 13, 119, 49, 3636, 695
1, 43, 40, 4, 375, 415, 21, 2, 92, 947, 19, 4, 2282, 1771, 14, 5, 106, 2, 115
1, 48, 25, 181, 8, 67, 6, 530, 9089, 1253, 7, 4, 2]),
       list([1, 14, 20, 16, 835, 835, 835, 51, 6, 1703, 56, 51, 6, 387, 180,
32, 812, 57, 2327, 6, 394, 437, 7, 676, 5, 58, 62, 24, 386, 12, 8, 61, 5301,
912, 37, 80, 106, 233]),
       list([1, 86, 125, 13, 62, 40, 8, 213, 46, 15, 137, 13, 244, 24, 35, 28
09, 4, 96, 4, 3100, 16, 2400, 80, 2384, 129, 1663, 4633, 4, 2, 115, 2085, 15,
2, 2, 165, 495, 9123, 18, 199, 4, 2, 88, 36, 70, 79, 35, 1271, 5, 4, 4824, 1
8, 24, 116, 23, 14, 17, 160, 2, 301, 2, 2799, 16, 2085, 9508, 4129, 36, 343,
3973, 17, 4, 2, 37, 47, 965, 602, 5, 60, 80, 2, 11, 4, 3100, 63, 9, 43, 379,
48, 4, 2619, 69, 1668, 90, 8, 2, 15, 96, 36, 62, 28, 839, 11, 294, 9954, 900,
36, 62, 30, 43, 2254, 18, 35, 1271, 2, 14, 506, 16, 115, 1803, 3383, 1204, 4
4, 4, 2, 34, 4, 2, 2, 1373, 7, 4, 1494, 525, 5, 60, 54, 1803, 34, 4, 4824, 33
83, 1204, 40, 54, 12, 645, 29, 100, 24, 1527, 48, 15, 218, 3793, 824, 13, 92,
124, 51, 9, 5, 6, 3275, 2408, 62, 28, 1840, 35, 2, 10, 10, 1324, 347, 12, 51
7, 125, 73, 19, 4, 2, 7, 112, 4, 4069, 7, 6, 506, 19, 2, 21, 4, 3100, 166, 1
4, 20, 5028, 1297]),
       ...,
       list([1, 13, 1408, 15, 8, 135, 14, 9, 35, 32, 46, 394, 20, 62, 30, 509
3, 21, 45, 184, 78, 4, 1492, 910, 769, 2290, 2515, 395, 4257, 5, 1454, 11, 11
9, 2, 89, 1036, 4, 116, 218, 78, 21, 407, 100, 30, 128, 262, 15, 7, 185, 228
0, 284, 1842, 2, 37, 315, 4, 226, 20, 272, 2942, 40, 29, 152, 60, 181, 8, 30,
50, 553, 362, 80, 119, 12, 21, 846, 5518]),
       list([1, 11, 119, 241, 9, 4, 840, 20, 12, 468, 15, 94, 3684, 562, 791,
39, 4, 86, 107, 8, 97, 14, 31, 33, 4, 2960, 7, 743, 46, 1028, 9, 3531, 5, 4,
768, 47, 8, 79, 90, 145, 164, 162, 50, 6, 501, 119, 7, 9, 4, 78, 232, 15, 16,
224, 11, 4, 333, 20, 4, 985, 200, 5, 2, 5, 9, 1861, 8, 79, 357, 4, 20, 47, 22
0, 57, 206, 139, 11, 12, 5, 55, 117, 212, 13, 1276, 92, 124, 51, 45, 1188, 7
1, 536, 13, 520, 14, 20, 6, 2302, 7, 470]),
       list([1, 6, 52, 7465, 430, 22, 9, 220, 2594, 8, 28, 2, 519, 3227, 6, 7
69, 15, 47, 6, 3482, 4067, 8, 114, 5, 33, 222, 31, 55, 184, 704, 5586, 2, 19,
346, 3153, 5, 6, 364, 350, 4, 184, 5586, 9, 133, 1810, 11, 5417, 2, 21, 4, 72
98, 2, 570, 50, 2005, 2643, 9, 6, 1249, 17, 6, 2, 2, 21, 17, 6, 1211, 232, 11
38, 2249, 29, 266, 56, 96, 346, 194, 308, 9, 194, 21, 29, 218, 1078, 19, 4, 7
8, 173, 7, 27, 2, 5698, 3406, 718, 2, 9, 6, 6907, 17, 210, 5, 3281, 5677, 47,
77, 395, 14, 172, 173, 18, 2740, 2931, 4517, 82, 127, 27, 173, 11, 6, 392, 21
7, 21, 50, 9, 57, 65, 12, 2, 53, 40, 35, 390, 7, 11, 4, 3567, 7, 4, 314, 74,
6, 792, 22, 2, 19, 714, 727, 5205, 382, 4, 91, 6533, 439, 19, 14, 20, 9, 144
1, 5805, 1118, 4, 756, 25, 124, 4, 31, 12, 16, 93, 804, 34, 2005, 2643])],
      dtype=object)
```

In [16]:
```
train_y
```

Out[16]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [17]:
```
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 (colu

#>>> import numpy as np
#>>> x = np.array((3,5,7))
#>>> y = np.array((5,7,9))
#>>> np.hstack((x,y))
# array([3, 5, 7, 5, 7, 9])


# You can see in the output above that the dataset is labeled into two categor
# which represents the sentiment of the review.

# The whole dataset contains 9998 unique words and the average review length i
```

```
Categories: [0 1]
Number of unique words: 9998
```

In [18]:
```
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 i
```

```
Average Review length: 234.75892
Standard Deviation: 173
```

In [19]:
```
# If you look at the data you will realize it has been already pre-processed.
# All words have been mapped to integers and the integers represent the words
# This is very common in text analysis to represent a dataset like this.
# So 4 represents the 4th most used word,
# 5 the 5th most used word and so on...
# The integer 1 is reserved for the start marker,
# the integer 2 for an unknown word and 0 for padding.

# Let's look at a single training example:

print("Label:", label[0])
```

```
Label: 1
```

In [20]:
```
print(data[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 3
6, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 17
2, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 19
2, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 1
6, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5,
62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130,
12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 2
8, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 376
6, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 8
8, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22,
21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4,
226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 53
45, 19, 178, 32]
```

In [21]:
```
# Let's decode the first review
# Above you see the first review of the dataset which is labeled as positive (
# The code below retrieves the dictionary mapping word indices back into the o
# It replaces every unknown word with a “#”. It does this by using the get_wor


# Retrieves a dict mapping words to their index in the IMDB dataset.
index = imdb.get_word_index()
# If there is a possibility of multiple keys with the same value, you will nee
# ivd = dict((v, k) for k, v in d.items())
# If you want to peek at the reviews yourself and see what people have actuall
reverse_index = dict([(value, key) for (key, value) in index.items()])
decoded = " ".join( [reverse_index.get(i - 3, "#") for i in data[0]] ) #The pu
print(decoded)
```

```
# this film was just brilliant casting location scenery story direction every
one's really suited the part they played and you could just imagine being the
re robert # is an amazing actor and now the same being director # father came
from the same scottish island as myself so i loved the fact there was a real
connection with this film the witty remarks throughout the film were great it
was just brilliant so much that i bought the film as soon as it was released
for # and would recommend it to everyone to watch and the fly fishing was ama
zing really cried at the end it was so sad and you know what they say if you
cry at a film it must have been good and this definitely was also # to the tw
o little boy's that played the # of norman and paul they were just brilliant
children are often left out of the # list i think because the stars that play
them all grown up are such a big profile for the whole film but these childre
n are amazing and should be praised for what they have done don't you think t
he whole story was so lovely because it was true and was someone's life after
all that was shared with us all
```

In [22]:
```
index
```

Out[22]:
```
{'fawn': 34701,
 'tsukino': 52006,
 'nunnery': 52007,
 'sonja': 16816,
 'vani': 63951,
 'woods': 1408,
 'spiders': 16115,
 'hanging': 2345,
 'woody': 2289,
 'trawling': 52008,
 "hold's": 52009,
 'comically': 11307,
 'localized': 40830,
 'disobeying': 30568,
 "'royale": 52010,
 "harpo's": 40831,
 'canet': 52011,
 'aileen': 19313,
 'acurately': 52012,
```

In [23]:
```
reverse_index
```

Out[23]:
```
{34701: 'fawn',
 52006: 'tsukino',
 52007: 'nunnery',
 16816: 'sonja',
 63951: 'vani',
 1408: 'woods',
 16115: 'spiders',
 2345: 'hanging',
 2289: 'woody',
 52008: 'trawling',
 52009: "hold's",
 11307: 'comically',
 40830: 'localized',
 30568: 'disobeying',
 52010: "'royale",
 40831: "harpo's",
 52011: 'canet',
 19313: 'aileen',
 52012: 'acurately',
```

In [24]:
```
decoded
```

Out[24]:
```
"# this film was just brilliant casting location scenery story direction ever
yone's really suited the part they played and you could just imagine being th
ere robert # is an amazing actor and now the same being director # father cam
e from the same scottish island as myself so i loved the fact there was a rea
l connection with this film the witty remarks throughout the film were great
it was just brilliant so much that i bought the film as soon as it was releas
ed for # and would recommend it to everyone to watch and the fly fishing was
amazing really cried at the end it was so sad and you know what they say if y
ou cry at a film it must have been good and this definitely was also # to the
two little boy's that played the # of norman and paul they were just brillian
t children are often left out of the # list i think because the stars that pl
ay them all grown up are such a big profile for the whole film but these chil
dren are amazing and should be praised for what they have done don't you thin
k the whole story was so lovely because it was true and was someone's life af
ter all that was shared with us all"
```

In [25]:
```
#Adding sequence to data
# Vectorization is the process of converting textual data into numerical vecto
data = vectorize(data)
label = np.array(label).astype("float32")

# Now it is time to prepare our data. We will vectorize every review and fill
# That means we fill every review that is shorter than 1000 with zeros.
# We do this because the biggest review is nearly that long and every input fo
# We also transform the targets into floats.
```

In [26]:
```
data
```

Out[26]:
```
array([[0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.],
       ...,
       [0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 1., ..., 0., 0., 0.]])
```

In [27]:
```
label
```

Out[27]:
```
array([1., 0., 0., ..., 0., 0., 0.], dtype=float32)
```

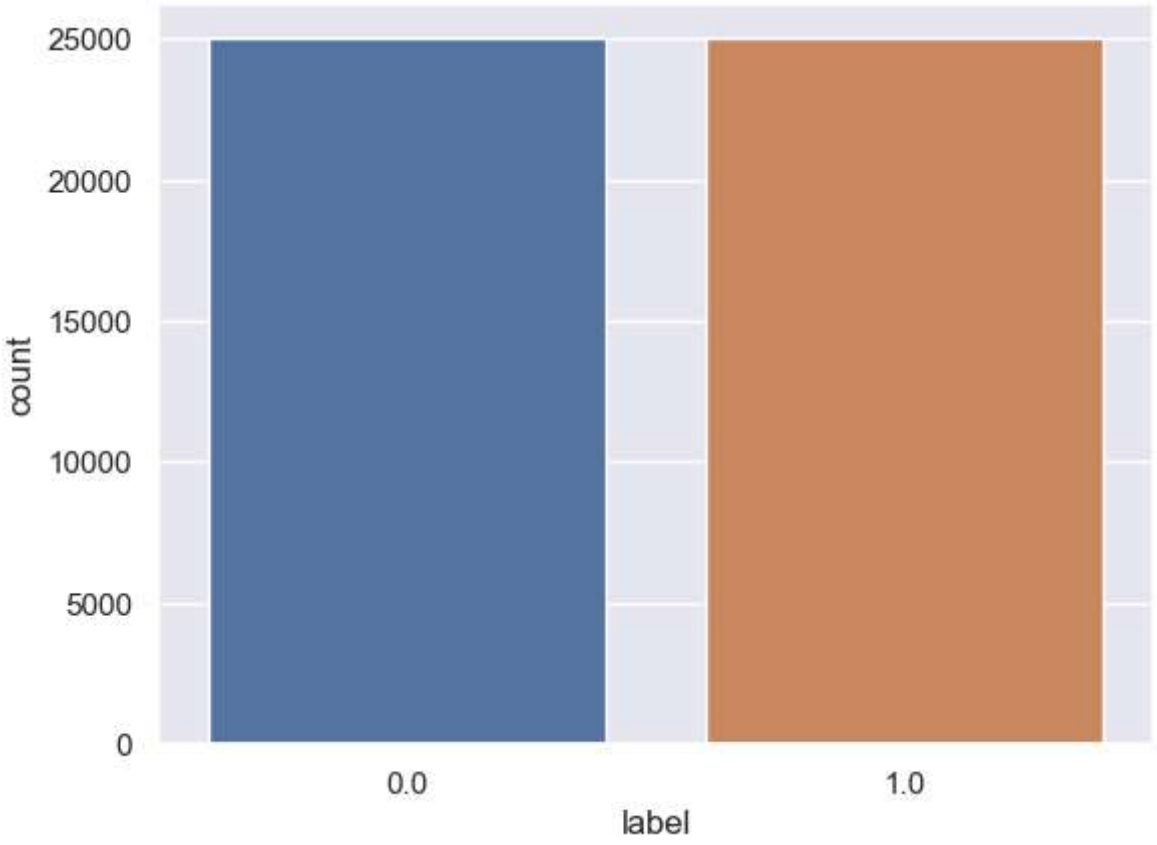
In [28]:
```
# Let's check distribution of data

# To create plots for EDA(exploratory data analysis)
import seaborn as sns #seaborn is a popular Python visualization library that
sns.set(color_codes=True)
import matplotlib.pyplot as plt # %matplotlib to display Matplotlib plots inli
%matplotlib inline
```

In [29]:
```
labelDF=pd.DataFrame({'label':label})
sns.countplot(x='label', data=labelDF)

# For below analysis it is clear that data has equel distribution of sentiment
```

Out[29]: <Axes: xlabel='label', ylabel='count'>



In [30]:
```
# 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
```

In [31]:
```
X_train.shape
```

Out[31]: (40000, 10000)

In [32]:
```
X_test.shape
```

Out[32]: (10000, 10000)

In [33]:
```
# Let's create sequential model,In deep learning, a Sequential model is a line

from keras.utils import to_categorical
from keras import models
from keras import layers
```

In [34]:
```
model = models.Sequential()
# Input - Layer
# Note that we set the input-shape to 10,000 at the input-layer because our re
# 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
# By the way, if you want you can build a sentiment analysis without LSTMs(Lon
model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu")) #ReLU" stands for Rectified L
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")) #adds another Dense layer t
model.summary()
```

```
WARNING:tensorflow:From C:\Users\Sakshi's PC\AppData\Roaming\Python\Python311
\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is dep
recated. Please use tf.compat.v1.get_default_graph instead.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                500050

 dropout (Dropout)           (None, 50)                0

 dense_1 (Dense)             (None, 50)                2550

 dropout_1 (Dropout)         (None, 50)                0

 dense_2 (Dense)             (None, 50)                2550

 dense_3 (Dense)             (None, 1)                 51

=================================================================
Total params: 505201 (1.93 MB)
Trainable params: 505201 (1.93 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [35]:
```
#For early stopping
# Stop training when a monitored metric has stopped improving.
# monitor: Quantity to be monitored.
# patience: Number of epochs with no improvement after which training will be
import tensorflow as tf #TensorFlow provides a wide range of tools and feature
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
```

In [36]:
```
# We use the “adam” optimizer, an algorithm that changes the weights and biase
# During training, the weights and biases of a machine learning model are upda
# We also choose binary-crossentropy as loss (because we deal with binary clas

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

```
WARNING:tensorflow:From C:\Users\Sakshi's PC\AppData\Roaming\Python\Python311
\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimi
zer is deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

In [37]:
```
# Now we're able to train our model. We'll do this with a batch_size of 500 an
# batch size defines the number of samples that will be propagated through the
# For instance, let's say you have 1050 training samples and you want to set u
# The algorithm takes the first 100 samples (from 1st to 100th) from the train
# Next, it takes the second 100 samples (from 101st to 200th) and trains the n
# We can keep doing this procedure until we have propagated all samples throug
# Problem might happen with the last set of samples. In our example, we've use
# The simplest solution is just to get the final 50 samples and train the netw
##The goal is to find the number of epochs that results in good performance on
results = model.fit(
 X_train, y_train,
 epochs= 2,
 batch_size = 500,
 validation_data = (X_test, y_test),
 callbacks=[callback]
)
```

```
Epoch 1/2
WARNING:tensorflow:From C:\Users\Sakshi's PC\AppData\Roaming\Python\Python311
\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTens
orValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue inste
ad.

WARNING:tensorflow:From C:\Users\Sakshi's PC\AppData\Roaming\Python\Python311
\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executin
g_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_
eagerly_outside_functions instead.

80/80 [==============================] - 5s 42ms/step - loss: 0.4081 - accura
cy: 0.8171 - val_loss: 0.2612 - val_accuracy: 0.8959
Epoch 2/2
80/80 [==============================] - 2s 19ms/step - loss: 0.2163 - accura
cy: 0.9172 - val_loss: 0.2542 - val_accuracy: 0.8964
```

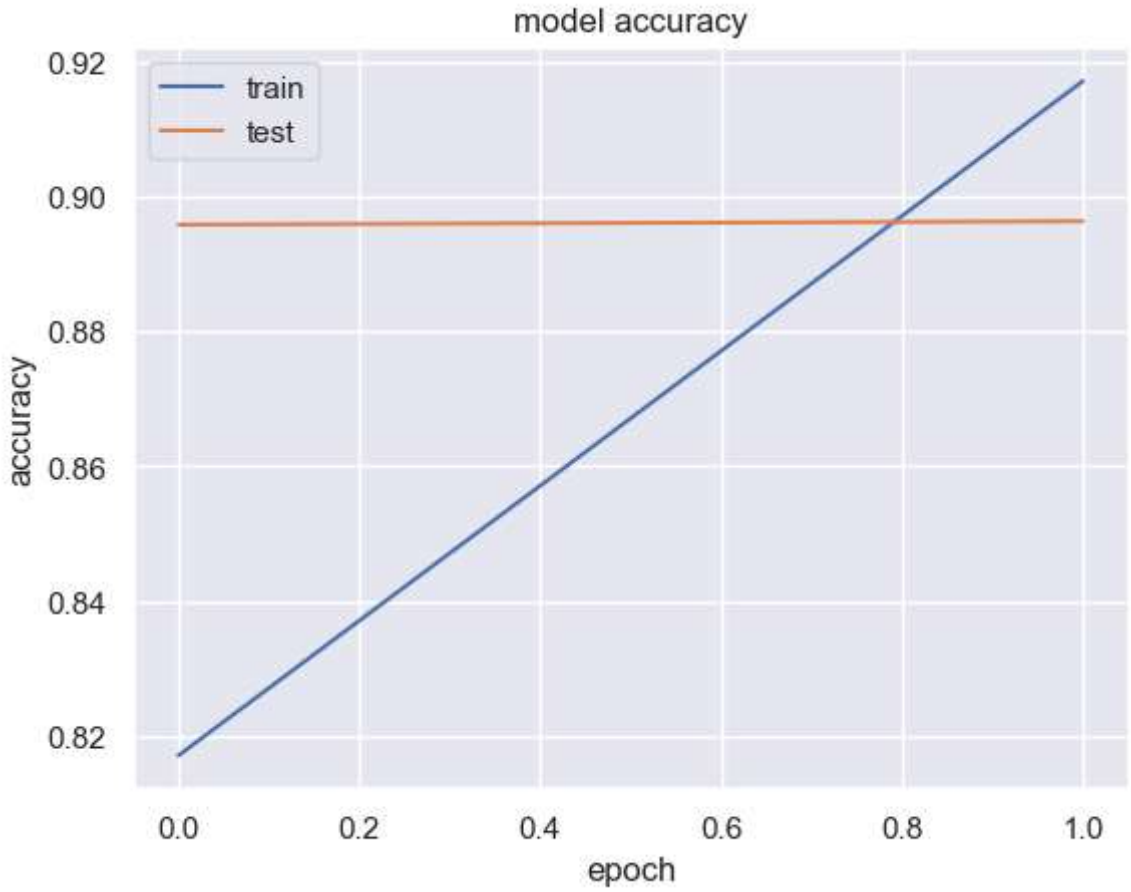
In [38]:
```
# Let's check mean accuracy of our model
print(np.mean(results.history["val_accuracy"])) # Good model should have a mea
```

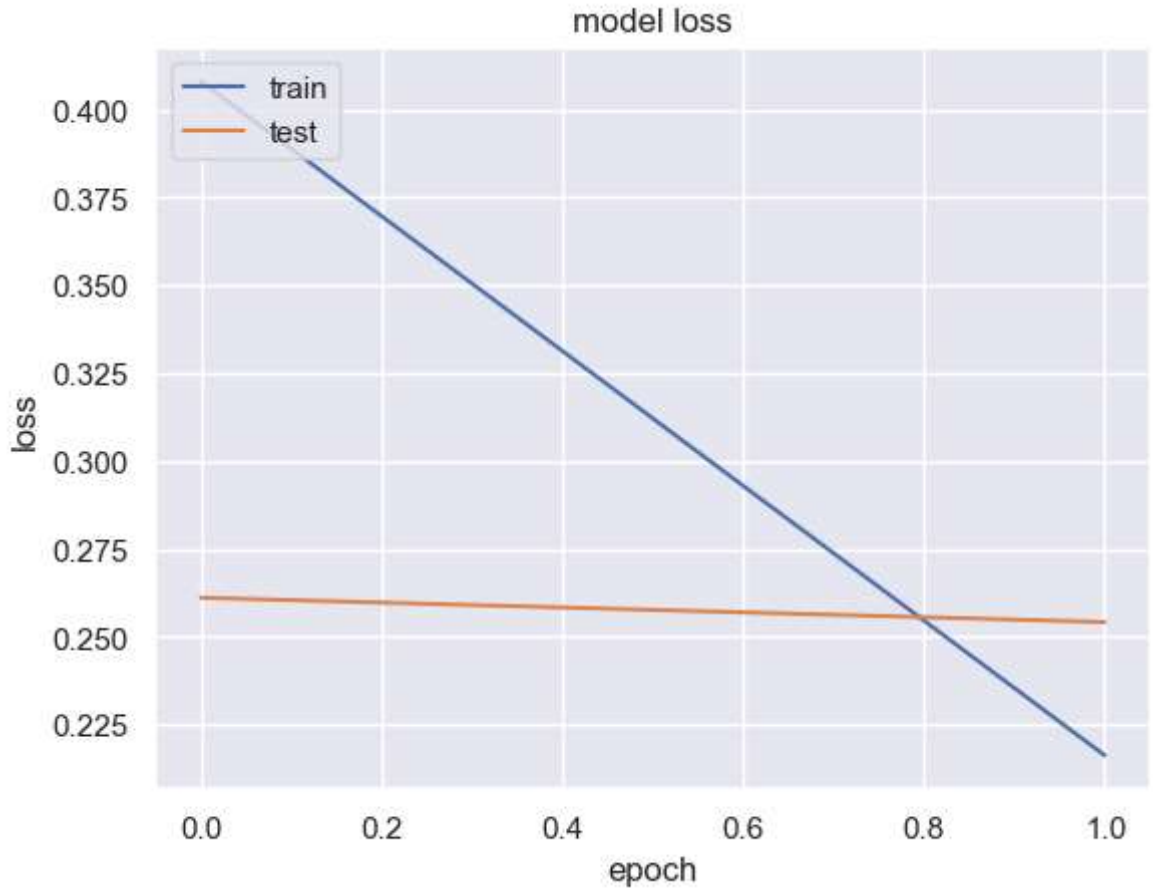
```
0.8961499929428101
```

In [39]:
```
#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']) #Plots the training accuracy of the mode
plt.plot(results.history['val_accuracy']) #Plots the validation accuracy of th
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'])
```





In [40]:
```
model.predict(X_test)
```

```
313/313 [==============================] - 1s 2ms/step
```

Out[40]:
```
array([[0.3707057 ],
       [0.97650874],
       [0.7507067 ],
       ...,
       [0.92500955],
       [0.971485  ],
       [0.98714596]], dtype=float32)
```

In [41]:
```
#Out put analysis,
#[0.9865479] is a single prediction value for a particular input sample in the
#It is the predicted probability of the positive sentiment class (class 1) for
#Since the output activation function of the last layer of the model is sigmoi
#,the output values represent the probabilities of the positive class.
#In this case, the probability of the positive class for the given input sampl
```