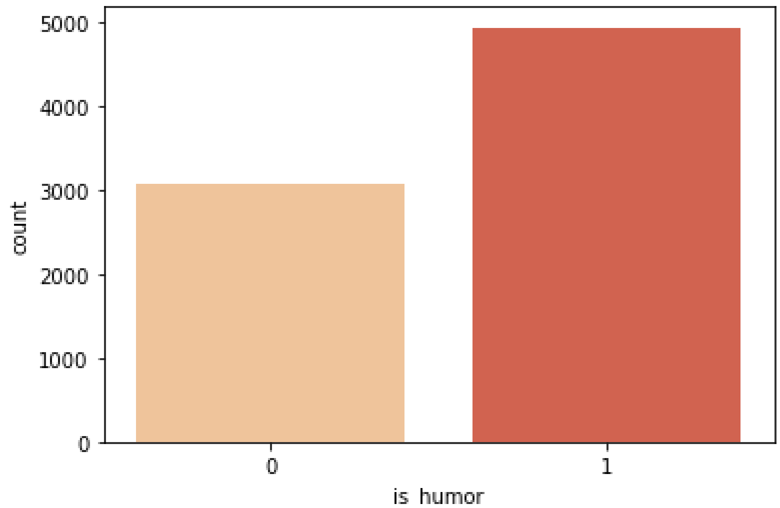
In [199]:
```
pip install wordcloud
```

```
Collecting wordcloud
  Downloading wordcloud-1.8.1-cp38-cp38-win_amd64.whl (155 kB)
Requirement already satisfied: pillow in c:\users\nisha\anaconda3\lib\site-
packages (from wordcloud) (7.2.0)
Requirement already satisfied: numpy>=1.6.1 in c:\users\nisha\anaconda3\lib
\site-packages (from wordcloud) (1.18.5)
Requirement already satisfied: matplotlib in c:\users\nisha\anaconda3\lib\s
ite-packages (from wordcloud) (3.2.2)
Requirement already satisfied: cycler>=0.10 in c:\users\nisha\anaconda3\lib
\site-packages (from matplotlib->wordcloud) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
c:\users\nisha\anaconda3\lib\site-packages (from matplotlib->wordcloud) (2.
4.7)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\nisha\anaco
nda3\lib\site-packages (from matplotlib->wordcloud) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\nisha\anaconda
3\lib\site-packages (from matplotlib->wordcloud) (1.2.0)
Requirement already satisfied: six in c:\users\nisha\anaconda3\lib\site-pac
kages (from cycler>=0.10->matplotlib->wordcloud) (1.15.0)
Installing collected packages: wordcloud
Successfully installed wordcloud-1.8.1
Note: you may need to restart the kernel to use updated packages.
```

In [200]:
```
import tensorflow as tf
import numpy as np
import pandas as pd
from wordcloud import WordCloud
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
```

In [202]:
```
#Distribution of humor values
import seaborn as sns
sns.countplot(x='is_humor',data=dataset, palette="OrRd")
```

Out[202]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2253cad2340>
```



In [201]:
```
dataset = pd.read_csv('train.csv')
X = dataset.iloc[:,1].values
#constant fill
from sklearn.impute import SimpleImputer
constant_imputer=SimpleImputer(strategy='constant', fill_value=0)
dataset.iloc[:]=constant_imputer.fit_transform(dataset)
print(X)

Y = dataset.iloc[:,2].values
train_examples, test_examples, train_labels, test_labels = train_test_split(X
print(Y)
```

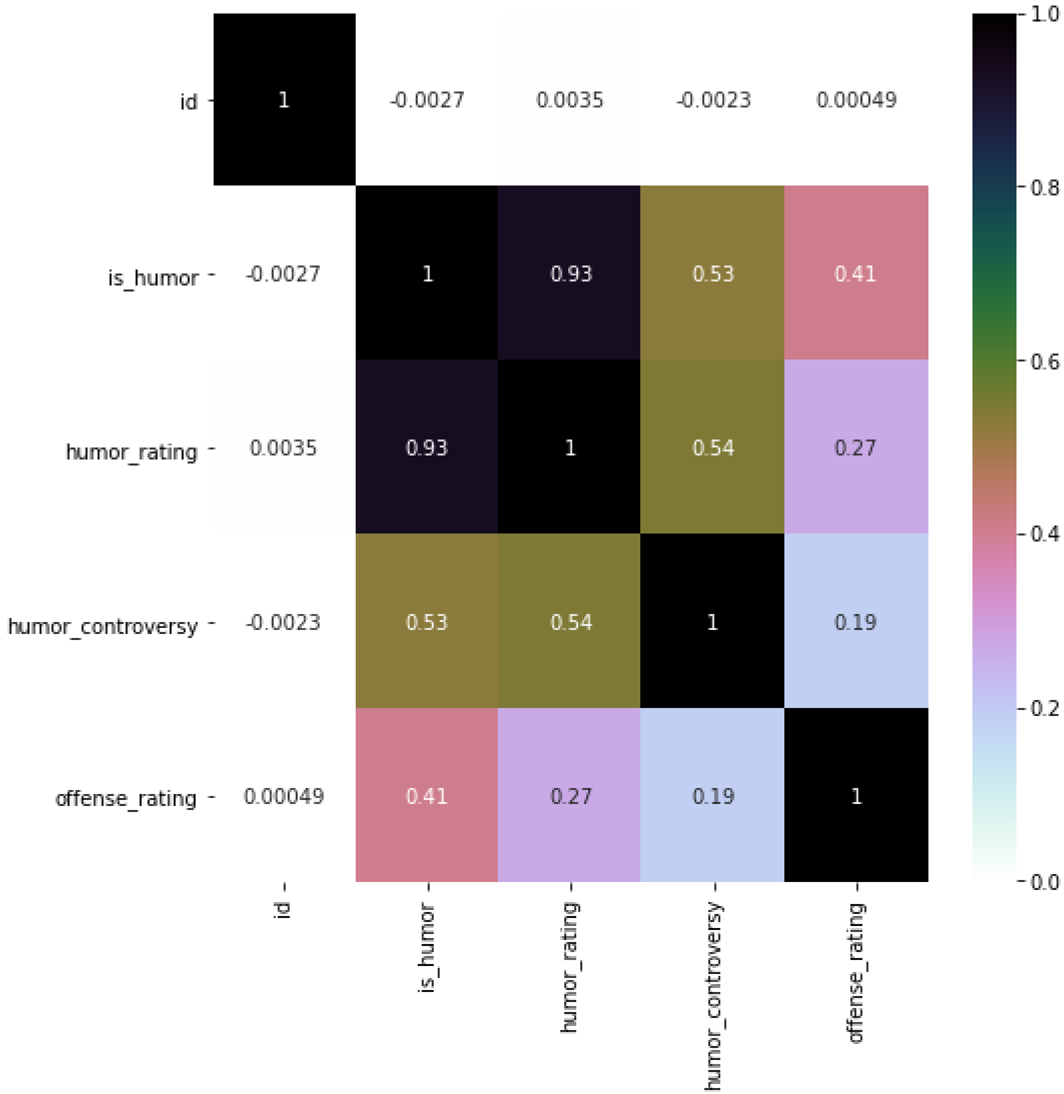
```
["TENNESSEE: We're the best state. Nobody even comes close. *Elevennessee w
alks into the room* TENNESSEE: Oh shit..."
 'A man inserted an advertisement in the classifieds "Wife Wanted". The nex
t day, he received 1000 of replies, all reading: "You can have mine." Free
delivery also available at your door step'
 'How many men does it take to open a can of beer? None. It should be open
by the time she brings it to the couch.'
 ...
 'Today, we Americans celebrate our independence from Britain while plannin
g our escape to Canada.'
 'How to keep the flies off the bride at an Italian wedding Keep a bucket o
f shit next to her'
 '"Each ounce of sunflower seeds gives you 37% of your daily need for vitam
in E" vitamin health']
[1 1 1 ... 1 1 0]
```

In [203]:
```
#Distribution of humor_controversy values
import seaborn as sns
sns.countplot(x='humor_controversy',data=dataset, palette="OrRd")
```

Out[203]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22541b4fdf0>
```

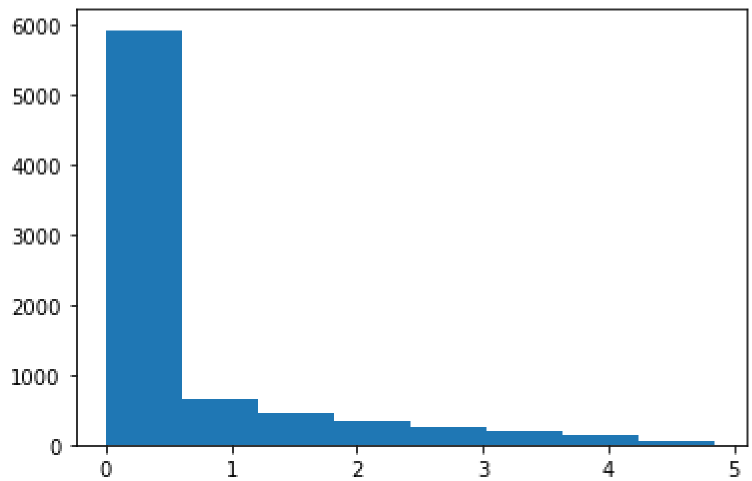
In [204]:
```
# Correlation of all the features in dataset
import seaborn as sns
plt.figure(figsize=(8,8))
sns.heatmap(dataset.corr(),annot=True,cmap='cubehelix_r')
```

Out[204]: <matplotlib.axes._subplots.AxesSubplot at 0x22500f03a60>



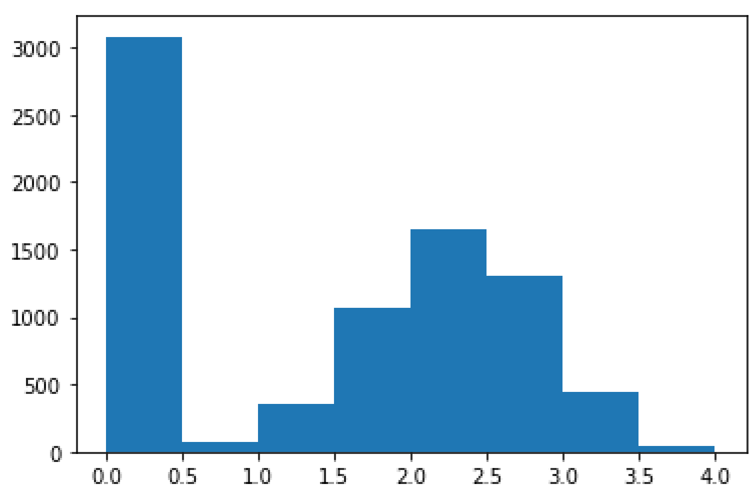
In [205]:

```
#Distribution of offense rating in dataset
import matplotlib.pyplot as plt

h = dataset.iloc[:,5].values
plt.hist(h, bins = 8)
plt.show()
```



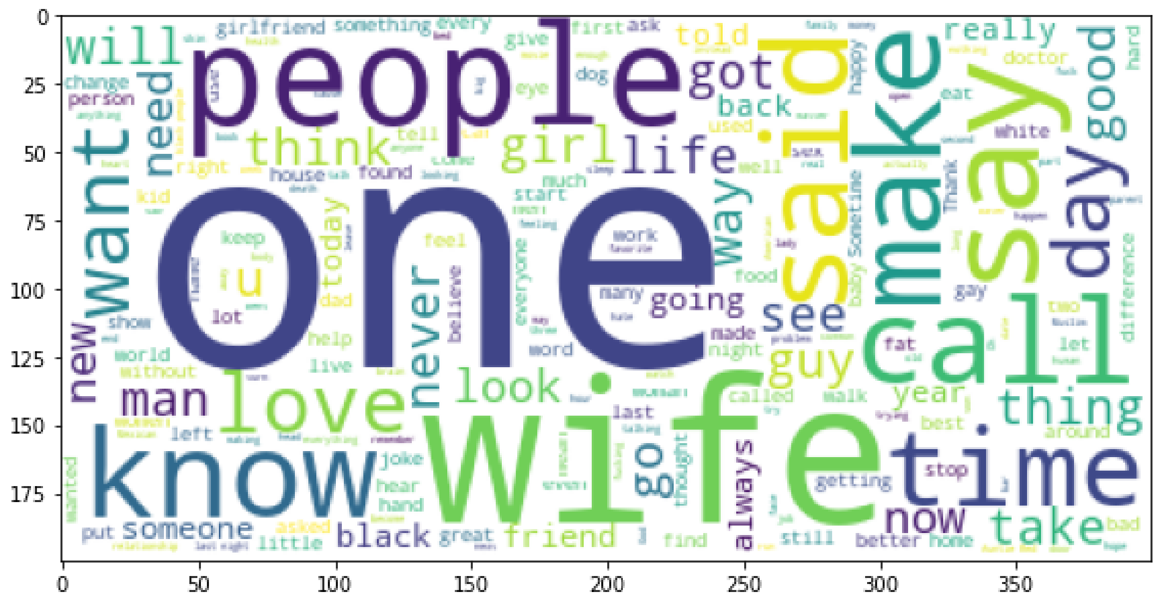
In [160]:

```
#Distribution of humor_rating in dataset
import matplotlib.pyplot as plt

h = dataset.iloc[:,3].values
plt.hist(h, bins = 8)
plt.show()
```



In [206]:

```
wc = WordCloud(background_color="white")
text = ''
for i in range (len(X)):
    text = text+ X[i]
plt.figure(figsize=(10,10))
wc.generate(text)
plt.axis("on")
plt.imshow(wc)
plt.show()
```



In [207]:

```
# Setting tokenizer properties
vocab_size = 50000
oov_tok = "<oov>"
# Fit the tokenizer on Training data
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(train_examples)
```

In [208]:

```
word_index = tokenizer.word_index
# Setting the padding properties
max_length = 500
trunc_type='post'
padding_type='post'
```

In [209]:

```
# Creating padded sequences from train and test data
training_sequences = tokenizer.texts_to_sequences(train_examples)
training_padded = pad_sequences(training_sequences, maxlen=max_length, paddir
testing_sequences = tokenizer.texts_to_sequences(test_examples)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length, padding=
```

In [210]:

```
# Setting the model parameters
embedding_dim = 300
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_ler
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid'),
    #tf.keras.layers.Dense(1, activation='linear')
])
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['accuracy']
model.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_18 (Embedding)     (None, 500, 300)          15000000  
_________________________________________________________________
global_average_pooling1d_18  (None, 300)               0         
_________________________________________________________________
dense_68 (Dense)             (None, 64)                19264     
_________________________________________________________________
dense_69 (Dense)             (None, 32)                2080      
_________________________________________________________________
dense_70 (Dense)             (None, 1)                 33        
=================================================================
Total params: 15,021,377
Trainable params: 15,021,377
Non-trainable params: 0
_________________________________________________________________
```

In [211]:

```
# Converting the lists to numpy arrays for Tensorflow 2.x
training_padded = np.array(training_padded)
training_labels = np.array(train_labels)
testing_padded = np.array(testing_padded)
testing_labels = np.array(test_labels)
```

In [212]:

```
print(len(training_padded))
```

```
6400
```

In [213]:

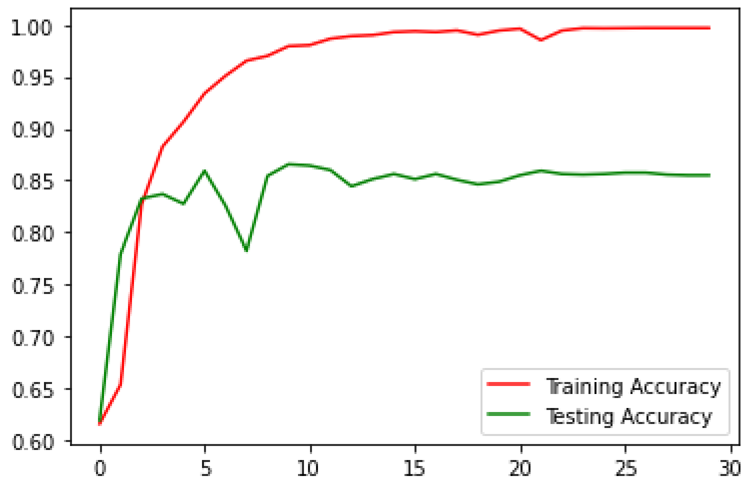
```
history = model.fit(training_padded, training_labels, epochs=30, validation_d
```

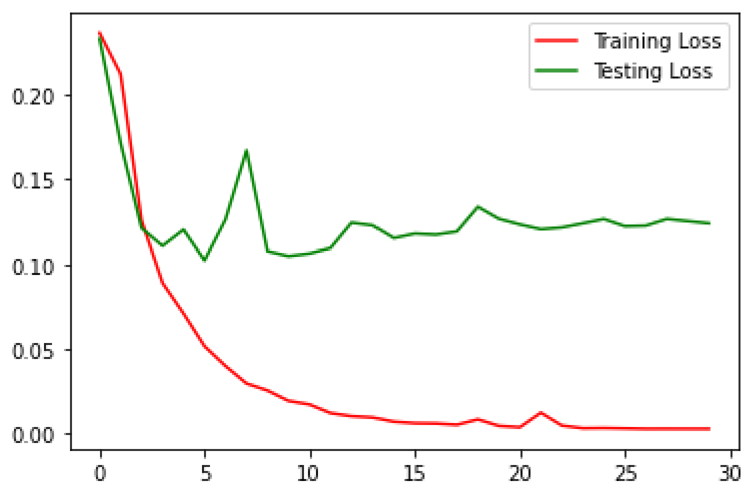
```
Epoch 1/30
200/200 [==============================] - 22s 111ms/step - loss: 0.2364
- accuracy: 0.6158 - val_loss: 0.2330 - val_accuracy: 0.6194
Epoch 2/30
200/200 [==============================] - 23s 117ms/step - loss: 0.2122
- accuracy: 0.6536 - val_loss: 0.1716 - val_accuracy: 0.7794
Epoch 3/30
200/200 [==============================] - 23s 114ms/step - loss: 0.1258
- accuracy: 0.8269 - val_loss: 0.1213 - val_accuracy: 0.8325
Epoch 4/30
200/200 [==============================] - 23s 115ms/step - loss: 0.0891
- accuracy: 0.8827 - val_loss: 0.1109 - val_accuracy: 0.8369
Epoch 5/30
200/200 [==============================] - 24s 121ms/step - loss: 0.0710
- accuracy: 0.9066 - val_loss: 0.1204 - val_accuracy: 0.8275
Epoch 6/30
200/200 [==============================] - 24s 119ms/step - loss: 0.0518
- accuracy: 0.9339 - val_loss: 0.1023 - val_accuracy: 0.8594
Epoch 7/30
200/200 [==============================] - 24s 119ms/step - loss: 0.0401
- accuracy: 0.9509 - val_loss: 0.1264 - val_accuracy: 0.8256
Epoch 8/30
200/200 [==============================] - 24s 118ms/step - loss: 0.0298
- accuracy: 0.9655 - val_loss: 0.1671 - val_accuracy: 0.7825
Epoch 9/30
200/200 [==============================] - 24s 120ms/step - loss: 0.0256
- accuracy: 0.9702 - val_loss: 0.1077 - val_accuracy: 0.8544
Epoch 10/30
200/200 [==============================] - 24s 120ms/step - loss: 0.0195
- accuracy: 0.9795 - val_loss: 0.1048 - val_accuracy: 0.8656
Epoch 11/30
200/200 [==============================] - 24s 121ms/step - loss: 0.0174
- accuracy: 0.9805 - val_loss: 0.1064 - val_accuracy: 0.8644
Epoch 12/30
200/200 [==============================] - 24s 120ms/step - loss: 0.0123
- accuracy: 0.9867 - val_loss: 0.1096 - val_accuracy: 0.8600
Epoch 13/30
200/200 [==============================] - 24s 120ms/step - loss: 0.0105
- accuracy: 0.9892 - val_loss: 0.1245 - val_accuracy: 0.8444
Epoch 14/30
200/200 [==============================] - 24s 122ms/step - loss: 0.0097
- accuracy: 0.9902 - val_loss: 0.1229 - val_accuracy: 0.8512
Epoch 15/30
200/200 [==============================] - 24s 121ms/step - loss: 0.0072
- accuracy: 0.9931 - val_loss: 0.1154 - val_accuracy: 0.8562
Epoch 16/30
200/200 [==============================] - 24s 121ms/step - loss: 0.0063
- accuracy: 0.9939 - val_loss: 0.1180 - val_accuracy: 0.8512
Epoch 17/30
200/200 [==============================] - 24s 121ms/step - loss: 0.0062
- accuracy: 0.9931 - val_loss: 0.1174 - val_accuracy: 0.8562
Epoch 18/30
200/200 [==============================] - 24s 121ms/step - loss: 0.0054
```

```
- accuracy: 0.9947 - val_loss: 0.1193 - val_accuracy: 0.8506
Epoch 19/30
200/200 [==============================] - 25s 123ms/step - loss: 0.0086
- accuracy: 0.9905 - val_loss: 0.1338 - val_accuracy: 0.8462
Epoch 20/30
200/200 [==============================] - 24s 121ms/step - loss: 0.0047
- accuracy: 0.9945 - val_loss: 0.1267 - val_accuracy: 0.8487
Epoch 21/30
200/200 [==============================] - 24s 122ms/step - loss: 0.0039
- accuracy: 0.9964 - val_loss: 0.1234 - val_accuracy: 0.8550
Epoch 22/30
200/200 [==============================] - 25s 124ms/step - loss: 0.0125
- accuracy: 0.9853 - val_loss: 0.1206 - val_accuracy: 0.8594
Epoch 23/30
200/200 [==============================] - 24s 122ms/step - loss: 0.0049
- accuracy: 0.9945 - val_loss: 0.1216 - val_accuracy: 0.8562
Epoch 24/30
200/200 [==============================] - 24s 121ms/step - loss: 0.0033
- accuracy: 0.9969 - val_loss: 0.1241 - val_accuracy: 0.8556
Epoch 25/30
200/200 [==============================] - 24s 122ms/step - loss: 0.0035
- accuracy: 0.9967 - val_loss: 0.1266 - val_accuracy: 0.8562
Epoch 26/30
200/200 [==============================] - 24s 122ms/step - loss: 0.0032
- accuracy: 0.9969 - val_loss: 0.1224 - val_accuracy: 0.8575
Epoch 27/30
200/200 [==============================] - 24s 121ms/step - loss: 0.0030
- accuracy: 0.9970 - val_loss: 0.1227 - val_accuracy: 0.8575
Epoch 28/30
200/200 [==============================] - 24s 122ms/step - loss: 0.0030
- accuracy: 0.9970 - val_loss: 0.1266 - val_accuracy: 0.8556
Epoch 29/30
200/200 [==============================] - 24s 122ms/step - loss: 0.0030
- accuracy: 0.9970 - val_loss: 0.1254 - val_accuracy: 0.8550
Epoch 30/30
200/200 [==============================] - 24s 122ms/step - loss: 0.0030
- accuracy: 0.9970 - val_loss: 0.1241 - val_accuracy: 0.8550
```

In [214]:

```
#Evaluating Accuracy and Loss of the model
%matplotlib inline
acc=history.history['accuracy']
val_acc=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']

epochs=range(len(acc)) #No. of epochs

#Plot training and validation accuracy per epoch
import matplotlib.pyplot as plt
plt.plot(epochs,acc,'r',label='Training Accuracy')
plt.plot(epochs,val_acc,'g',label='Testing Accuracy')
plt.legend()
plt.figure()

#Plot training and validation loss per epoch
plt.plot(epochs,loss,'r',label='Training Loss')
plt.plot(epochs,val_loss,'g',label='Testing Loss')
plt.legend()
plt.show()
```





In [215]:

```
codalab_data = pd.read_csv('public_dev.csv')
test_data_codalab = codalab_data['text'].values
test_data_codalab
```

```
       'Today is National Orgasm Day? What is the world cumming to?',
       'Every morning, I do 100 pushups and 300 crunches, then follow it
 up with 2 huge lies about my morning routine.',
       "Teach a man to fish? Never. It's hard enough to catch those slipp
ery devils without the competition. I'll teach a man a scary story about
 how the ocean will kill you if you even think about fishing.",
       'I am, regrettably, here',
       "Give an African a fish and he will eat for a day. Teach an Africa
n to phish and he'll steal your identity.",
       'I almost got raped in jail. My family takes monopoly way too seri
ously.',

       'Having just seen some SAG awards pics, it should be noted that Me
ryl can get it',
       'A great way to distract citizens from the disintegration of democ
ratic norms = alienate them and so thoroughly from the democratic process
that they stop paying attention altogether',
       "A little fact about me: 'I can hold my pee all night' was my leas
t successful pickup line.",
       'I dunno, I don’t trust anything about this!!! I’m broken and so i
```

In [ ]:

```
#for i in range(len(test_data_codalab)):
 #   print(test_data_codalab[i])
  #  test_data_codalab[i] = remove_stopwords(test_data_codalab[i].lower())
   # print(test_data_codalab[i])
```

In [217]:

```
sequences = tokenizer.texts_to_sequences(test_data_codalab)
# print(sequences)
padded = pad_sequences(sequences, maxlen=max_length, padding=padding_type, tr
humor = model.predict_classes(padded)
for i in range(len(humor)):
    print(humor[i][0])
```

```
1
1
1
1
1
1
1
0
1
0
0
1
0
0
1
1
1
1
1
0
0
1
1
1
1
1
0
0
1
1
1
1
1
0
1
1
1
1
0
1
1
1
0
1
1
1
0
1
0
1
```

```
0
1
1
0
1
0
1
1
1
0
1
0
1
1
0
0
0
0
0
1
0
1
1
0
0
1
0
1
1
0
0
0
1
0
0
1
1
1
1
1
1
0
0
1
0
1
0
1
1
0
1
1
1
0
1
1
1
```

```
1
1
0
1
0
1
0
1
0
0
1
0
0
1
1
1
0
1
0
0
1
1
0
1
1
1
1
0
0
1
1
0
1
0
1
1
0
1
0
1
0
1
0
1
1
1
0
1
1
0
0
0
0
0
1
0
1
```

```
1
1
1
0
1
1
0
1
1
1
0
0
0
0
1
0
0
0
0
1
1
1
1
0
0
1
1
1
1
1
1
0
1
0
1
1
1
0
0
0
1
0
1
1
1
1
1
1
0
1
1
0
0
1
0
1
1
```

```
1
1
1
1
0
0
1
1
0
1
1
1
1
1
1
1
1
1
0
1
1
1
0
1
1
0
1
1
0
0
1
1
1
0
1
1
0
0
1
1
1
1
1
1
0
1
1
1
1
1
1
0
1
1
1
1
1
```

```
0
1
0
0
1
0
1
1
1
0
1
0
0
0
1
1
1
1
0
1
0
0
0
0
1
0
1
1
1
0
1
0
0
1
1
0
1
0
0
1
0
1
1
1
1
0
1
1
0
1
1
1
0
0
1
1
1
```

```
1
1
1
0
1
0
0
0
1
1
0
0
0
1
1
0
1
1
1
1
0
1
1
0
0
1
1
0
1
0
0
0
1
0
1
1
0
0
0
1
1
1
0
0
1
1
1
1
1
0
0
0
1
1
1
0
0
```

```
0
1
1
0
1
0
1
1
0
1
1
1
1
0
1
1
1
0
1
1
1
1
1
1
1
0
1
1
1
0
0
0
1
0
1
0
0
1
1
1
1
1
1
1
1
1
1
1
0
0
1
0
0
1
1
1
1
```

```
1
1
0
0
1
1
0
1
1
0
1
1
0
1
1
1
0
1
1
1
0
1
0
0
0
1
0
1
1
1
1
1
0
0
1
0
1
1
1
0
0
1
1
1
1
1
1
1
0
0
1
1
1
1
0
1
1
```

```
0
1
1
1
1
1
0
1
0
0
0
1
0
1
0
1
0
1
0
0
0
0
1
0
1
1
0
1
1
1
1
0
0
1
1
0
1
1
0
1
1
1
0
1
1
1
1
1
1
1
1
0
1
1
0
0
0
```

```
1
1
0
0
1
0
1
1
1
1
1
1
0
1
0
1
1
1
1
1
1
1
1
0
0
0
0
1
1
0
1
1
1
0
1
1
1
1
1
0
1
0
1
1
1
0
0
1
1
1
1
0
1
0
0
1
1
```

```
1
1
0
1
0
0
1
1
1
1
1
1
1
1
1
1
1
1
0
1
0
0
0
1
1
1
1
0
1
1
1
0
1
0
1
1
1
0
1
0
1
0
1
1
1
1
1
1
1
1
1
1
1
1
0
1
1
```

```
1
0
1
1
0
0
1
0
1
1
0
1
1
1
0
0
1
0
1
1
1
1
0
1
1
1
0
0
1
1
1
1
0
1
1
1
1
1
1
0
1
1
1
1
1
0
0
0
1
0
1
0
1
1
1
1
0
```

```
1
1
1
0
0
1
1
1
1
1
1
1
1
1
1
1
0
1
1
0
1
0
1
1
0
0
1
1
1
1
0
0
0
1
1
0
0
1
1
0
0
1
1
1
1
0
0
0
0
1
1
1
1
0
0
1
0
```

```
1
0
1
1
0
1
1
1
1
1
0
1
1
1
0
0
0
1
0
1
1
0
0
1
1
1
1
0
0
0
1
1
1
0
1
1
1
0
1
0
0
1
0
1
1
1
0
1
1
0
1
0
0
1
1
0
1
```

```
1
1
0
0
0
0
0
1
1
1
0
1
1
1
1
1
0
0
0
0
0
1
0
0
0
1
0
0
1
0
0
1
0
1
0
0
0
1
0
1
1
1
0
1
1
1
1
1
0
1
1
1
0
1
1
0
0
```

```
1
1
1
0
0
0
1
1
1
1
0
1
0
0
1
1
1
1
0
0
0
0
1
0
0
0
1
1
1
1
1
1
1
0
1
1
1
1
0
0
1
0
1
1
0
1
0
0
0
1
1
1
0
1
1
1
1
```

```
0
1
1
0
0
1
1
1
1
0
1
1
1
0
1
1
1
1
1
1
0
1
0
1
1
1
1
0
0
1
0
1
1
1
0
1
1
1
```