```
df = pd.read_csv('data.csv',delimiter=',',encoding='latin-1')
df = df[['Category','Message']]
df = df[pd.notnull(df['Message'])]
df.rename(columns = {'Message':'Message'}, inplace = True)
df.head()
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
df['Category'].value_counts()
```

```
ham     4825
spam     747
Name: Category, dtype: int64
```

```
spam = df[df['Message'].str.contains("win" and "free")]
spam['Category'].value_counts()
```

```
spam    61
ham     61
Name: Category, dtype: int64
```

```
ham_message_length = []
spam_message_length = []
for i in df.values:
    if(i[0] == "ham"):
        ham_message_length.append(len(i[1]))
    else:
        spam_message_length.append(len(i[1]))
ham_message_length
```

```
23,
58,
37,
61,
60,
54,
69,
72,
40,
36,
35,
34,
```

```
        78,
        121,
        64,
        25,
        22,
        35,
        156,
        58,
        31,
        24,
        17,
        152,
        41,
        80,
        85,
        115,
        25,
        45,
        70,
        22,
        43,
        22,
        22,
        72,
        91,
        70,
        140,
        31,
        52,
        40,
        23,
        145,
        54,
        43,
        30,
        39,
        71,
        146,
        73,
        23,
        26,
        59,
        150,
        166,
        26,
        ...]
```

```
import pandas as pd
from gensim.models.word2vec import Word2Vec
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.layers import Dense, Dropout, Conv1D, MaxPool1D, GlobalMaxPool1D, Embedding, Activ
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
```

```
from keras.models import Sequential
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.snowball import PorterStemmer
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np


def preprocess_text(sen):
    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sen)

    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

    # Removing multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)

    stops = stopwords.words('english')
    #print(stops)
    porter = PorterStemmer()
    for word in sentence.split():
        if word in stops:
            sentence = sentence.replace(word, '')
        sentence = sentence.replace(word, porter.stem(word))
    return sentence.lower()
df['Message'] = df['Message'].apply(preprocess_text)


mes = []
for i in df['Message']:
    mes.append(i.split())
print(mes[:2])
```

```
[['go', 'jurong', 'pot', 'crazi', 'avail', 'bugi', 'great', 'world', 'la', 'buffet', 'c
```

```
word2vec_model = Word2Vec(mes, size=500, window=3, min_count=1, workers=16)
print(word2vec_model)
```

```
WARNING:gensim.models.base_any2vec:under 10 jobs per worker: consider setting a smaller
Word2Vec(vocab=7259, size=500, alpha=0.025)
```

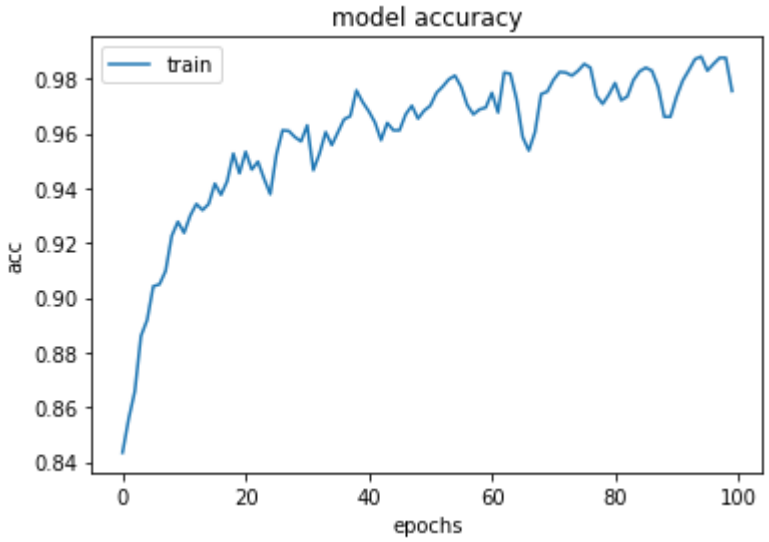
```
token = Tokenizer(7229)
token.fit_on_texts(df['Message'])
```

```
text = token.texts_to_sequences(df['Message'])
text = pad_sequences(text, 75)

le = preprocessing.LabelEncoder()
y = le.fit_transform(df['Category'])
y = to_categorical(y)


X_train, X_test, y_train, y_test = train_test_split(np.array(text), y, test_size=0.2, stratif


import tensorflow as tf
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(units = 110,activation = 'relu'))
model.add(tf.keras.layers.Dense(units = 110,activation = 'relu'))
model.add(tf.keras.layers.Dense(units = 2,activation = 'sigmoid'))
model.compile(optimizer = 'adam', loss = 'binary_crossentropy' , metrics = ['accuracy'])
history = model.fit(X_train,y_train,batch_size = 32, epochs = 100)
```

```
140/140 [==============================] - 0s 1ms/step - loss: 0.0989 - accuracy: 0.
Epoch 73/100
140/140 [==============================] - 0s 2ms/step - loss: 0.0925 - accuracy: 0.
Epoch 74/100
140/140 [==============================] - 0s 1ms/step - loss: 0.1042 - accuracy: 0.
Epoch 75/100
140/140 [==============================] - 0s 1ms/step - loss: 0.0831 - accuracy: 0.
Epoch 76/100
140/140 [==============================] - 0s 2ms/step - loss: 0.0715 - accuracy: 0.
Epoch 77/100
140/140 [==============================] - 0s 2ms/step - loss: 0.1003 - accuracy: 0.
Epoch 78/100
140/140 [==============================] - 0s 2ms/step - loss: 0.1536 - accuracy: 0.
Epoch 79/100
140/140 [==============================] - 0s 2ms/step - loss: 0.2090 - accuracy: 0.
Epoch 80/100
140/140 [==============================] - 0s 2ms/step - loss: 0.1728 - accuracy: 0.
Epoch 81/100
140/140 [==============================] - 0s 2ms/step - loss: 0.1148 - accuracy: 0.
Epoch 82/100
140/140 [==============================] - 0s 2ms/step - loss: 0.1689 - accuracy: 0.
Epoch 83/100
140/140 [==============================] - 0s 2ms/step - loss: 0.1268 - accuracy: 0.
Epoch 84/100
140/140 [==============================] - 0s 1ms/step - loss: 0.1147 - accuracy: 0.
Epoch 85/100
140/140 [==============================] - 0s 2ms/step - loss: 0.0904 - accuracy: 0.
Epoch 86/100
140/140 [==============================] - 0s 2ms/step - loss: 0.0797 - accuracy: 0.
Epoch 87/100
140/140 [==============================] - 0s 2ms/step - loss: 0.0817 - accuracy: 0.
Epoch 88/100
140/140 [==============================] - 0s 2ms/step - loss: 0.1248 - accuracy: 0.
Epoch 89/100
140/140 [==============================] - 0s 2ms/step - loss: 0.2308 - accuracy: 0.
Epoch 90/100
140/140 [==============================] - 0s 1ms/step - loss: 0.1704 - accuracy: 0.
Epoch 91/100
```

```
140/140 [==============================] - 0s 1ms/step - loss: 0.1360 - accuracy: 0.
Epoch 92/100
140/140 [==============================] - 0s 1ms/step - loss: 0.0949 - accuracy: 0.
Epoch 93/100
140/140 [==============================] - 0s 2ms/step - loss: 0.0707 - accuracy: 0.
Epoch 94/100
140/140 [==============================] - 0s 1ms/step - loss: 0.0476 - accuracy: 0.
Epoch 95/100
140/140 [==============================] - 0s 1ms/step - loss: 0.0546 - accuracy: 0.
Epoch 96/100
140/140 [==============================] - 0s 2ms/step - loss: 0.0816 - accuracy: 0.
Epoch 97/100
140/140 [==============================] - 0s 1ms/step - loss: 0.0592 - accuracy: 0.
Epoch 98/100
140/140 [==============================] - 0s 1ms/step - loss: 0.0591 - accuracy: 0.
Epoch 99/100
140/140 [==============================] - 0s 2ms/step - loss: 0.0605 - accuracy: 0.
Epoch 100/100
140/140 [==============================] - 0s 1ms/step - loss: 0.1280 - accuracy: 0.
```

```
plt.plot(history.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('acc')
plt.xlabel('epochs')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.savefig('model_accuracy.png')

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




```
from tkinter.constants import X
message = ['Congratulations! you have won a $1,000 Walmart gift card. Go to http://bit.ly/123
seq = token.texts_to_sequences(message)

padded = pad_sequences(seq, maxlen=text.shape[1], dtype='int32', value=0)

pred = model.predict(padded)

labels = ['ham','spam']
print(pred, labels[np.argmax(pred)])
```

```
1/1 [==============================] - 0s 54ms/step
[[9.9999821e-01 3.1830814e-06]] ham
```

```
message2 = ['thanks for accepting my request to connect']
seq2 = token.texts_to_sequences(message2)

padded = pad_sequences(seq2, maxlen=text.shape[1], dtype='int32', value=0)

pred = model.predict(padded)

labels = ['ham','spam']
print(pred, labels[np.argmax(pred)])
```

```
1/1 [==============================] - 0s 12ms/step
[[9.9999571e-01 1.1309023e-05]] ham
```