**Assignment - 4**

# SMS Spam Classification

| | |
|---|---|
| **Assignment Date** | **08 October 2022** |
| **Student Name** | **Meena Kaveri R** |
| **Student Roll Number** | **2019504549** |
| **Maximum Marks** | **2 Marks** |

**Problem Statement:**

**Problem Statement:**

Over recent years, as the popularity of mobile phone devices has increased, Short Message Service (SMS) has grown into a multi-billion dollar industry. At the same time, reduction in the cost of messaging services has resulted in growth in unsolicited commercial advertisements (spams) being sent to mobile phones. Due to Spam SMS, Mobile service providers suffer from some sort of financial problems as well as it reduces calling time for users. Unfortunately, if the user accesses such Spam SMS they may face the problem of virus or malware. When SMS arrives at mobile it will disturb mobile user privacy and concentration. It may lead to frustration for the user. So Spam SMS is one of the major issues in the wireless communication world and it grows day by day.

## Solution: **Spam Message Classification using LSTM**

**Source code and corresponding outputs:**

## 1.Import the Necessary Libraries

In [2]:

```
import numpy as np
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/sms-spam-collection-dataset/spam.csv
```

## 2. Reading the .csv dataset

In [3]:
```
data=pd.read_csv("../input/sms-spam-collection-dataset/spam.csv",encoding="latin")
data.head()
```

Out[3]:

| | v1 | v2 | Unnamed: 2 | Unnamed: 3 | Unnamed: 4 |
|---|---|---|---|---|---|
| 0 | ham | Go until jurong point, crazy.. Available only ... | NaN | NaN | NaN |
| 1 | ham | Ok lar... Joking wif u oni... | NaN | NaN | NaN |
| 2 | spam | Free entry in 2 a wkly comp to win FA Cup fina... | NaN | NaN | NaN |
| 3 | ham | U dun say so early hor... U c already then say... | NaN | NaN | NaN |
| 4 | ham | Nah I don't think he goes to usf, he lives aro... | NaN | NaN | NaN |

In [4]:
```
data.columns
```

Out[4]:
```
Index(['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], dtype='object')
```

## 3. Drop the unnamed Columns

In [5]:
```
data=data.drop(columns=["Unnamed: 2","Unnamed: 3","Unnamed: 4"])
```

## 4. Renaming Column names sensible

In [6]:
```
data=data.rename(
{
    "v1":"Category",
    "v2":"Message"
},
    axis=1
)
```

In [7]:
```
data.head()
```

Out[7]:

| | 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... |

## 5. Check for null values in dataset

In [8]:
```
data.isnull().sum()
```

Out[8]:
```
Category    0
Message     0
dtype: int64
```

In [9]:
```
data.info()
```

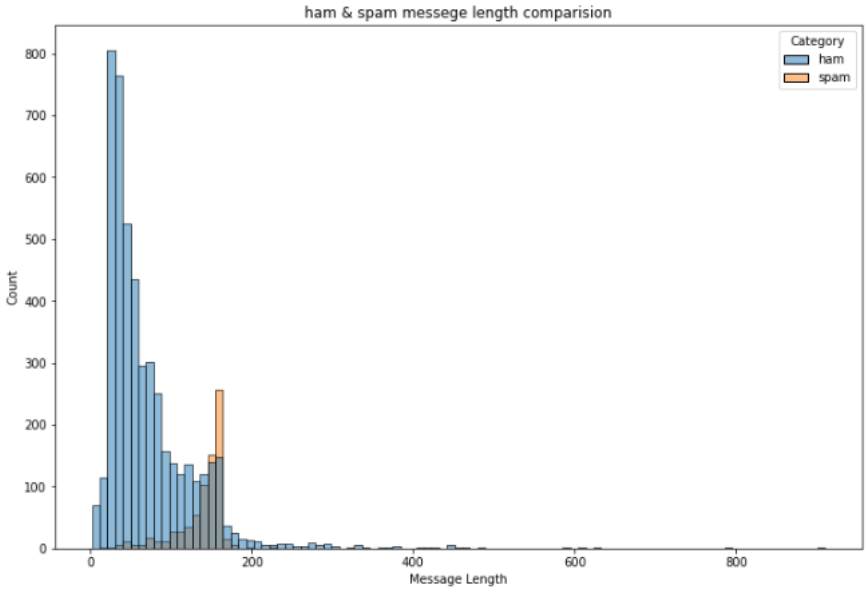
```
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Category  5572 non-null   object
 1   Message   5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

## 6.Creating a new Field to store the Message Lengths

In [10]:
```
data["Message Length"]=data["Message"].apply(len)
```

## 7. Histogram Inference of Message Lengths of Spam and Non-spam messages

In [11]:
```
fig=plt.figure(figsize=(12,8))
sns.histplot(
    x=data["Message Length"],
    hue=data["Category"]
)
plt.title("ham & spam messege length comparision")
plt.show()
```



In [12]:

```
ham_desc=data[data["Category"]=="ham"]["Message Length"].describe()
spam_desc=data[data["Category"]=="spam"]["Message Length"].describe()

print("Ham Messege Length Description:\n",ham_desc)
print("**********************************")
print("Spam Message Length Description:\n",spam_desc)
```

```
Ham Messege Length Description:
 count    4825.000000
mean       71.023627
std        58.016023
min         2.000000
25%        33.000000
50%        52.000000
75%        92.000000
max       910.000000
Name: Message Length, dtype: float64
**********************************
Spam Message Length Description:
 count    747.000000
mean     138.866131
std       29.183082
min       13.000000
25%      132.500000
50%      149.000000
75%      157.000000
max      224.000000
Name: Message Length, dtype: float64
```

In [13]:

```
data.describe(include="all")
```

Out[13]:

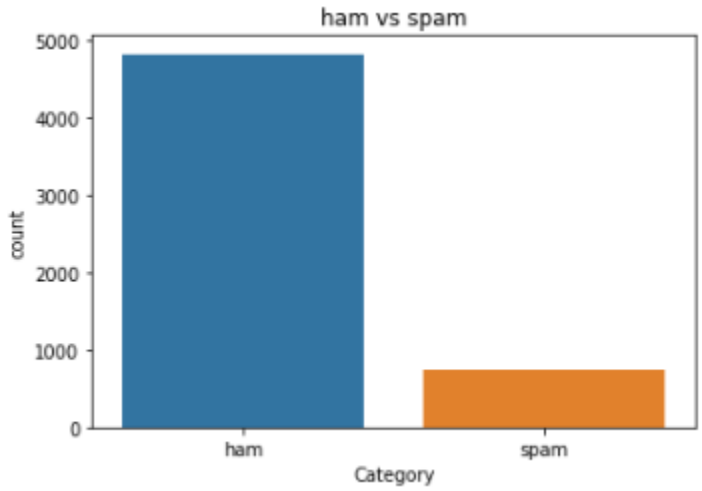
| | Category | Message | Message Length |
|---|---|---|---|
| **count** | 5572 | 5572 | 5572.000000 |
| **unique** | 2 | 5169 | NaN |
| **top** | ham | Sorry, I'll call later | NaN |
| **freq** | 4825 | 30 | NaN |
| **mean** | NaN | NaN | 80.118808 |
| **std** | NaN | NaN | 59.690841 |
| **min** | NaN | NaN | 2.000000 |
| **25%** | NaN | NaN | 36.000000 |
| **50%** | NaN | NaN | 61.000000 |
| **75%** | NaN | NaN | 121.000000 |
| **max** | NaN | NaN | 910.000000 |

## 8. Visualizing count of messages of Spam and Non Spam

In [14]:
```
data["Category"].value_counts()
```

Out[14]:
```
ham     4825
spam     747
Name: Category, dtype: int64
```

In [15]:
```
sns.countplot(
    data=data,
    x="Category"
)
plt.title("ham vs spam")
plt.show()
```




In [16]:
```
ham_count=data["Category"].value_counts()[0]
spam_count=data["Category"].value_counts()[1]

total_count=data.shape[0]

print("Ham contains:{:.2f}% of total data.".format(ham_count/total_count*100))
print("Spam contains:{:.2f}% of total data.".format(spam_count/total_count*100))
```

```
Ham contains:86.59% of total data.
Spam contains:13.41% of total data.
```

## 9. Undersampling to Genralize Model and Balance Spam and Ham quantities in dataset

In [17]:
```
minority_len=len(data[data["Category"]=="spam"])
majority_len=len(data[data["Category"]=="ham"])
minority_indices=data[data["Category"]=="spam"].index
majority_indices=data[data["Category"]=="ham"].index
random_majority_indices=np.random.choice(
    majority_indices,
    size=minority_len,
    replace=False
)

undersampled_indices=np.concatenate([minority_indices,random_majority_indices])
df=data.loc[undersampled_indices]
df=df.sample(frac=1)
df=df.reset_index()
df=df.drop(
    columns=["index"],
)
```

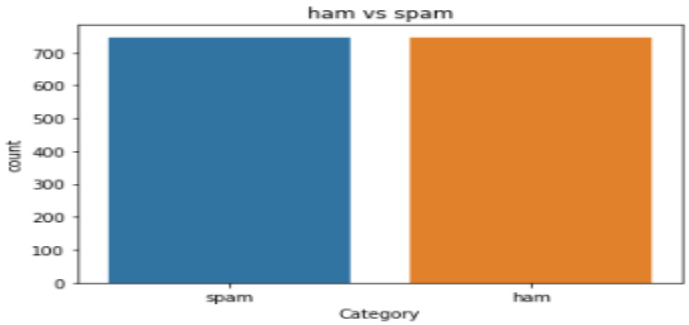
In [18]:
```
df.shape
```

Out[18]: (1494, 3)

In [19]:
```
df["Category"].value_counts()
```

Out[19]:
```
ham     747
spam    747
Name: Category, dtype: int64
```

In [20]:
```
sns.countplot(
    data=df,
    x="Category"
)
plt.title("ham vs spam")
plt.show()
```




Display the head of new **df**

In [21]:
```
df.head()
```

Out[21]:

| | Category | Message | Message Length |
|---|---|---|---|
| 0 | spam | FREE>Ringtone! Reply REAL or POLY eg REAL1 1. ... | 158 |
| 1 | spam | URGENT! We are trying to contact U Todays draw... | 157 |
| 2 | ham | Ok ill send you with in <DECIMAL> ok. | 45 |
| 3 | ham | Oh just getting even with u.... u? | 34 |
| 4 | spam | A link to your picture has been sent. You can ... | 96 |

## 10. Binary Encoding of Spam and Ham Categories

In [22]:
```
df["Label"]=df["Category"].map(
    {
        "ham":0,
        "spam":1
    }
)
```

In [23]:
```
df.head()
```

Out[23]:

| | Category | Message | Message Length | Label |
|---|---|---|---|---|
| 0 | spam | FREE>Ringtone! Reply REAL or POLY eg REAL1 1. ... | 158 | 1 |
| 1 | spam | URGENT! We are trying to contact U Todays draw... | 157 | 1 |
| 2 | ham | Ok ill send you with in <DECIMAL> ok. | 45 | 0 |
| 3 | ham | Oh just getting even with u.... u? | 34 | 0 |
| 4 | spam | A link to your picture has been sent. You can ... | 96 | 1 |

## 11. Import Necessary Libraries to perform Word Tokenization

In [24]:
```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

stemmer=PorterStemmer()
```

In [25]:
```
corpus=[]
for message in df["Message"]:
    message=re.sub("[^a-zA-Z]"," ",message)
    message=message.lower()
    message=message.split()
    message=[stemmer.stem(words)
             for words in message
              if words not in set(stopwords.words("english"))
             ]
    message=" ".join(message)
    corpus.append(message)
```

## 12. Perform One Hot on Corpus

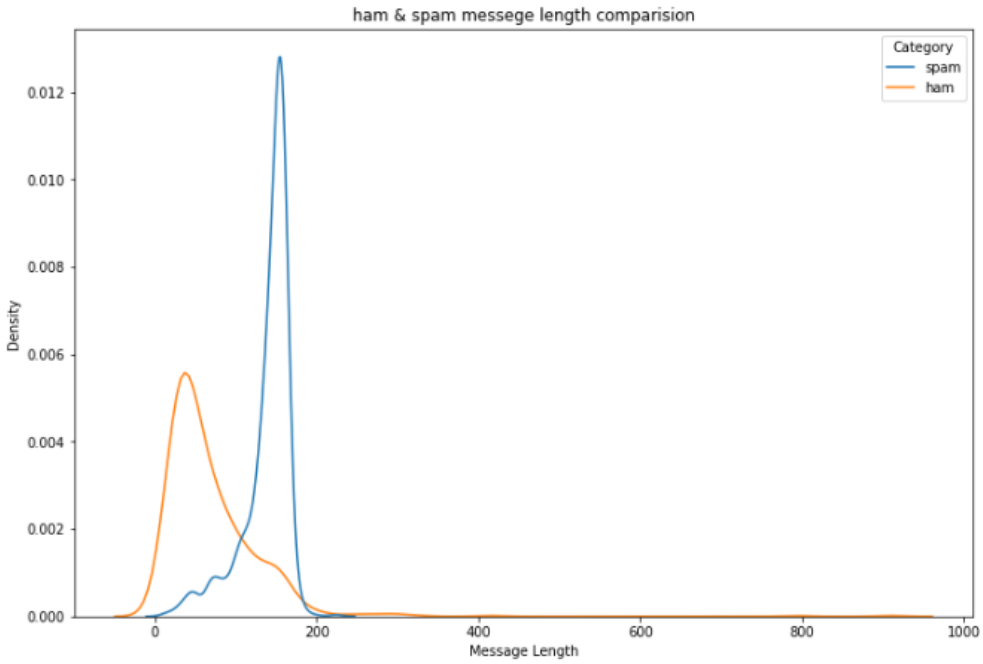
In [26]:
```
from tensorflow.keras.preprocessing.text import one_hot
vocab_size=10000

oneHot_doc=[one_hot(words,n=vocab_size)
           for words in corpus
           ]
```

In [27]:
```
df["Message Length"].describe()
```

Out[27]:
```
count    1494.000000
mean      104.491299
std        60.362332
min         2.000000
25%        49.000000
50%       114.000000
75%       153.000000
max       910.000000
Name: Message Length, dtype: float64
```

In [28]:
```
fig=plt.figure(figsize=(12,8))
sns.kdeplot(
    x=df["Message Length"],
    hue=df["Category"]
)
plt.title("ham & spam messege length comparision")
plt.show()
```



In [29]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
sentence_len=200
embedded_doc=pad_sequences(
    oneHot_doc,
    maxlen=sentence_len,
    padding="pre"
)
```

In [30]:
```
extract_features=pd.DataFrame(
    data=embedded_doc
)
target=df["Label"]
```

In [31]:
```
df_final=pd.concat([extract_features,target],axis=1)
```

In [32]:
```
df_final.head()
```

Out[32]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 191 | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | Label |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 8116 | 8983 | 7883 | 1884 | 5957 | 5877 | 266 | 1527 | 5846 | 1 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 9989 | 7682 | 5710 | 5519 | 2447 | 1240 | 3994 | 6950 | 3655 | 1 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 3310 | 6099 | 7761 | 9276 | 4679 | 2205 | 3310 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 8194 | 7945 | 3841 | 266 | 266 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 5677 | 7440 | 8481 | 9975 | 2366 | 4841 | 4320 | 4320 | 4672 | 1 |

5 rows × 201 columns

## 13. Splitting Dependent and Independent Variables

In [33]:
```
X=df_final.drop("Label",axis=1)
y=df_final["Label"]
```

## 14. Train, test and Validation Split

In [34]:
```
from sklearn.model_selection import train_test_split
```

In [35]:
```
X_trainval,X_test,y_trainval,y_test=train_test_split(
    X,
    y,
    random_state=42,
    test_size=0.15
)
```

In [36]:
```
X_train,X_val,y_train,y_val=train_test_split(
    X_trainval,
    y_trainval,
    random_state=42,
    test_size=0.15
)
```

## 15.Building a Sequential Model

In [37]:
```
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Embedding
from tensorflow.keras.models import Sequential
```

In [38]:
```
model=Sequential()
```

In [39]:
```
feature_num=100
model.add(
    Embedding(
        input_dim=vocab_size,
        output_dim=feature_num,
        input_length=sentence_len
    )
)
model.add(
    LSTM(
    units=128
    )
)

model.add(
    Dense(
        units=1,
        activation="sigmoid"
    )
)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 200, 100)          1000000
_________________________________________________________________
lstm (LSTM)                  (None, 128)               117248
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 1,117,377
Trainable params: 1,117,377
Non-trainable params: 0
_________________________________________________________________
```

In [40]:
```
from tensorflow.keras.optimizers import Adam
model.compile(
    optimizer=Adam(
    learning_rate=0.001
    ),
    loss="binary_crossentropy",
    metrics=["accuracy"]
)
```

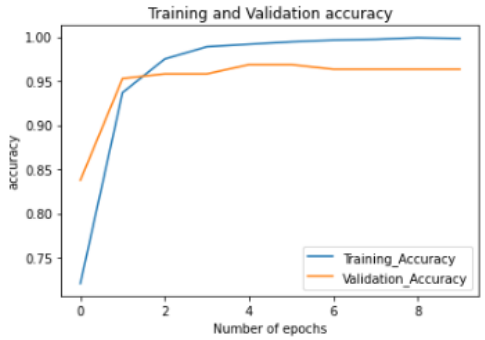
# 16. Model Fitting

In [41]:
```
history=model.fit(
    X_train,
    y_train,
    validation_data=(
        X_val,
        y_val
    ),
    epochs=10
)
```

```
Epoch 1/10
34/34 [==============================] - 24s 633ms/step - loss: 0.6324 - accuracy: 0.6331 - val_loss: 0.4218 - val_accuracy: 0.8377
Epoch 2/10
34/34 [==============================] - 21s 608ms/step - loss: 0.3045 - accuracy: 0.9257 - val_loss: 0.1631 - val_accuracy: 0.9529
Epoch 3/10
34/34 [==============================] - 21s 609ms/step - loss: 0.1046 - accuracy: 0.9689 - val_loss: 0.1231 - val_accuracy: 0.9581
Epoch 4/10
34/34 [==============================] - 21s 621ms/step - loss: 0.0465 - accuracy: 0.9880 - val_loss: 0.1293 - val_accuracy: 0.9581
Epoch 5/10
34/34 [==============================] - 21s 613ms/step - loss: 0.0342 - accuracy: 0.9895 - val_loss: 0.1252 - val_accuracy: 0.9686
Epoch 6/10
34/34 [==============================] - 21s 615ms/step - loss: 0.0179 - accuracy: 0.9951 - val_loss: 0.1366 - val_accuracy: 0.9686
Epoch 7/10
34/34 [==============================] - 21s 614ms/step - loss: 0.0121 - accuracy: 0.9968 - val_loss: 0.1314 - val_accuracy: 0.9634
Epoch 8/10
34/34 [==============================] - 21s 619ms/step - loss: 0.0222 - accuracy: 0.9944 - val_loss: 0.1479 - val_accuracy: 0.9634
Epoch 9/10
34/34 [==============================] - 21s 614ms/step - loss: 0.0077 - accuracy: 0.9989 - val_loss: 0.1624 - val_accuracy: 0.9634
Epoch 10/10
34/34 [==============================] - 21s 614ms/step - loss: 0.0077 - accuracy: 0.9976 - val_loss: 0.1751 - val_accuracy: 0.9634
```

In [42]:
```
metrics = pd.DataFrame(history.history)
metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy': 'Training_Accuracy', 'val_loss': 'Validation_Loss', 'val_accuracy': 'Validation_Accurac
def plot_graph_acc(var1, var2, string):
    metrics[[var1, var2]].plot()
    plt.title('Training and Validation ' + string)
    plt.xlabel ('Number of epochs')
    plt.ylabel(string)
    plt.legend([var1, var2])
```

In [43]:
```
plot_graph_acc('Training_Accuracy', 'Validation_Accuracy', 'accuracy')
```



## 17. Save and Test the Model

In [44]:
```
y_pred=model.predict(X_test)
y_pred=(y_pred>0.5)
```

In [45]:
```
model.save('Spam_SMS_classifier.h5')
```

## 18. Evaluating the Model

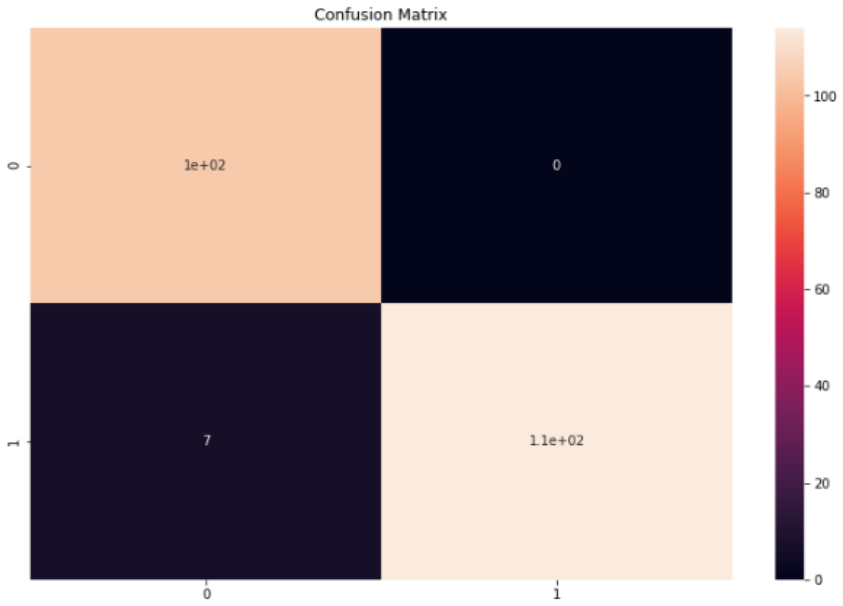
In [46]:
```
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [47]:
```
score=accuracy_score(y_test,y_pred)
print("Test Score:{:.2f}%".format(score*100))
```

```
Test Score:96.89%
```

In [48]:
```
cm=confusion_matrix(y_test,y_pred)
fig=plt.figure(figsize=(12,8))
sns.heatmap(
    cm,
    annot=True,
)
plt.title("Confusion Matrix")
cm
```

Out[48]:
```
array([[104,   0],
       [  7, 114]])
```



## 19. Function to Test the Model on a Random message

In [62]:
```
def classify_message(model,message):
    for sentences in message:
        sentences=nltk.sent_tokenize(message)
        for sentence in sentences:
            words=re.sub("[^a-zA-Z]"," ",sentence)
            if words not in set(stopwords.words('english')):
                word=nltk.word_tokenize(words)
                word=" ".join(word)
    oneHot=[one_hot(word,n=vocab_size)]
    text=pad_sequences(oneHot,maxlen=sentence_len,padding="pre")
    predict=model.predict(text)
    if predict>0.5:
        print("It is a spam")
        print("predict score: ", predict[0][0])
    else:
        print("It is not a spam")
        print("predict score: ", predict[0][0])
```

In [80]:
```
message1="I am having my Tests right now. Will call back as soon as possible! Till then be safe wherever you are. Be Alert of any hazard"
message2="Your Rs.8850 welcome bonus is ready to be credited. Download Junglee Rummy now. Claim Bonus on your first deposit prize pool"
```

In [81]:
```
classify_message(model,message1)
```

```
It is not a spam
predict score:  0.037389785
```

In [82]:
```
classify_message(model,message2)
```

```
It is a spam
predict score:  0.9936712
```