Assignment-4 (SMS SPAM Classification)

```
    import numpy as np import pandas as pd import seaborn as sns
   import matplotlib.pyplot as plt
```

```
data =
pd.read_csv('/content/sample_data/spam.csv',delimiter=',',encoding='la
tin-1') data.head()
```

```
     v1                                                   v2 Unnamed: 2
\
0    ham  Go until jurong point, crazy.. Available only ...        NaN

1    ham                       Ok lar... Joking wif u oni...        NaN

2    spam  Free entry in 2 a wkly comp to win FA Cup fina...        NaN

3    ham  U dun say so early hor... U c already then say...        NaN

4    ham  Nah I don't think he goes to usf, he lives aro...        NaN


  Unnamed: 3 Unnamed: 4   0
NaN         NaN
1         NaN         NaN
2         NaN         NaN
3         NaN         NaN  4         NaN         NaN  data.columns
```

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

```
#drop the unamed columns
data=data.drop(columns=["Unnamed: 2","Unnamed: 3","Unnamed: 4"])
```

```
#rename the two relevant columns
data=data.rename(
{
    "v1":"Category",
    "v2":"Message"
},axis=1)
data.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...
```

```
#check for null values
data.isnull().sum()
```

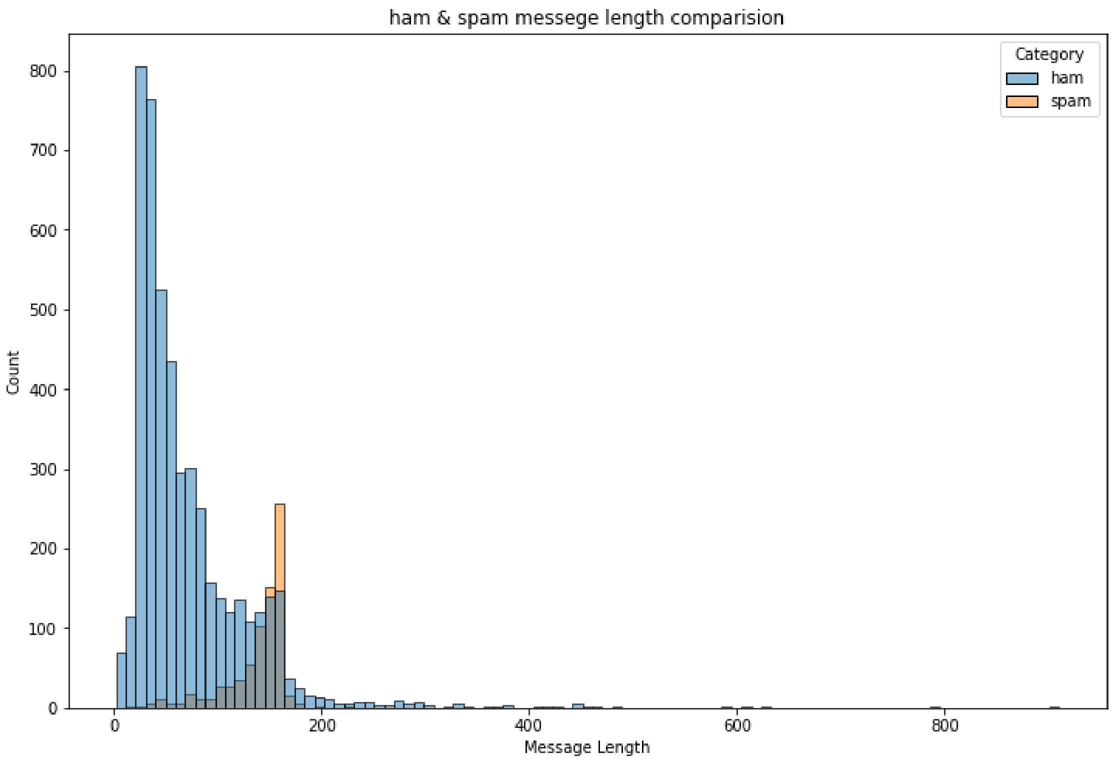
```
Category    0
Message     0
dtype: int64
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
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
```

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

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



```
#Display the description of length of ham and spam messages seperately
on an individual series.

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
```

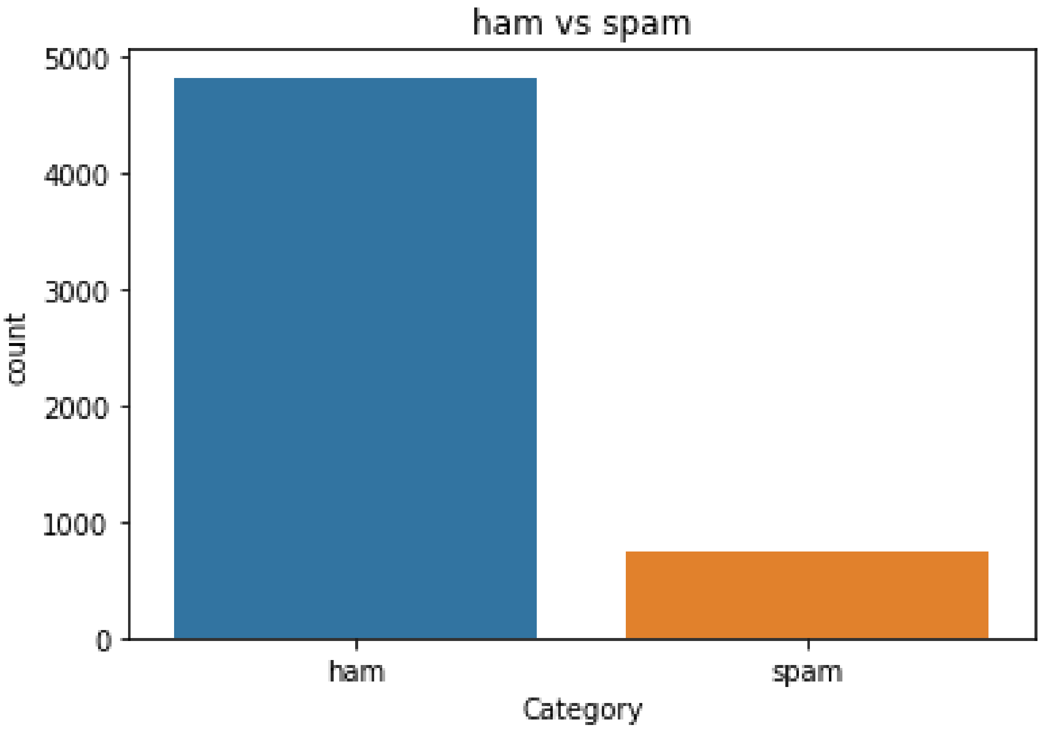
```
data.describe(include="all")
```

```
       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
```

```
data["Category"].value_counts()
```

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

```
sns.countplot(
data=data,
x="Category"
)
plt.title("ham vs spam")
plt.show()
```



```
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.
```

```
#compute the length of majority & minority class
minority_len=len(data[data["Category"]=="spam"])
majority_len=len(data[data["Category"]=="ham"])

#store the indices of majority and minority class
minority_indices=data[data["Category"]=="spam"].index
majority_indices=data[data["Category"]=="ham"].index

#generate new majority indices from the total majority_indices
#with size equal to minority class length so we obtain equivalent
number of indices length
random_majority_indices=np.random.choice(
majority_indices,     size=minority_len,
    replace=False
)

#concatenate the two indices to obtain indices of new dataframe
undersampled_indices=np.concatenate([minority_indices,random_majority_
indices])

#create df using new indices
df=data.loc[undersampled_indices]

#shuffle the sample
df=df.sample(frac=1)

#reset the index as its all mixed
df=df.reset_index()

#drop the older index
df=df.drop(
    columns=["index"],
)
```

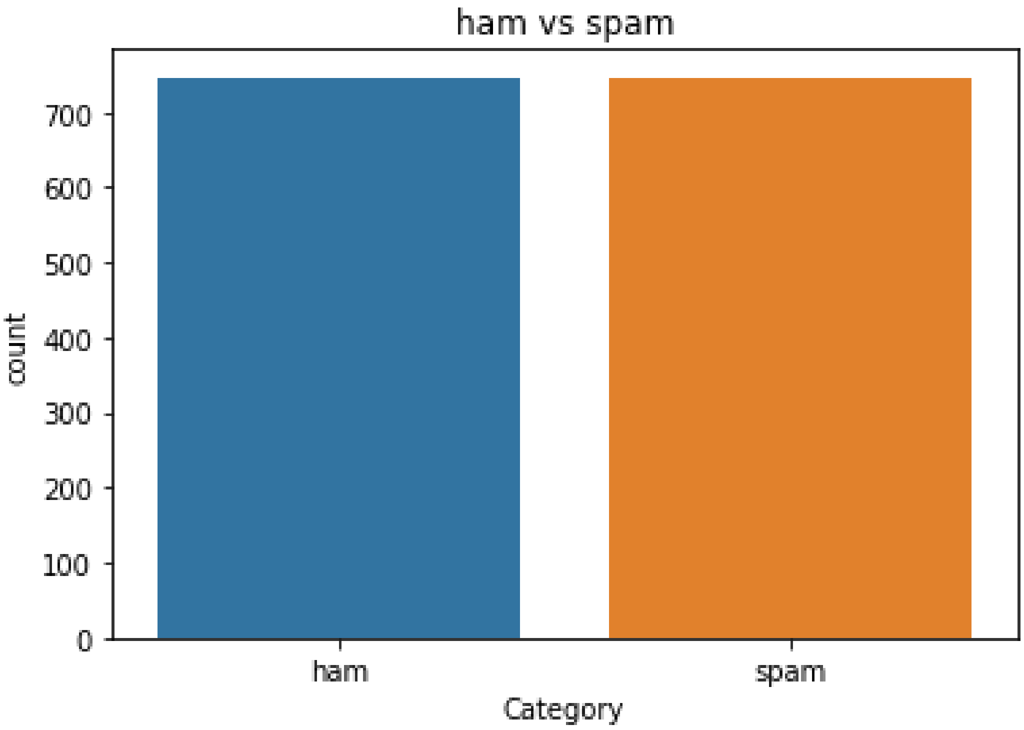
```
df.shape
```

```
(1494, 3)
```

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

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

```
sns.countplot(
data=df,
x="Category"
)
plt.title("ham vs spam")
plt.show()
```



```
df.head()
```

```
  Category                                               Message  Message
Length
0      ham  Aah! A cuddle would be lush! I'd need lots of ...
87
1      ham             I'm in solihull, | do you want anything?
40
2      spam  Double Mins & 1000 txts on Orange tariffs. Lat...
       151
3      ham  No we put party 7 days a week and study lightl...
       126
4      spam  URGENT!! Your 4* Costa Del Sol Holiday or å£50...

161
```

```
#Created new column Label and encode ham as 0 and spam as 1
df["Label"]=df["Category"].map(
    {
        "ham":0,
"spam":1
    }
)
```

```
df.head()
```

```
  Category                                               Message  Message
Length  \
0      ham  Aah! A cuddle would be lush! I'd need lots of ...
       87
1      ham           I'm in solihull, | do you want anything?
40
2      spam  Double Mins & 1000 txts on Orange tariffs. Lat...
151
3      ham  No we put party 7 days a week and study lightl...
126
4      spam  URGENT!! Your 4* Costa Del Sol Holiday or å£50...
       161

   Label
0      0
1      0
2      1
3      0   4      1
```

```
import re
import nltk
from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

stemmer=PorterStemmer()

nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip. True
```

```
#declare empty list to store tokenized message
corpus=[]

#iterate through the df["Message"]
for message in df["Message"]:

    #replace every special characters, numbers etc.. with whitespace
of message
    #It will help retain only letter/alphabets
message=re.sub("[^a-zA-Z]"," ",message)

    #convert every letters to its lowercase
message=message.lower()

    #split the word into individual word list
message=message.split()

    #perform stemming using PorterStemmer for all non-englishstopwords
    message=[stemmer.stem(words)
            for words in message
```

```
             if words not in set(stopwords.words("english"))
            ]
    #join the word lists with the whitespace
message=" ".join(message)

    #append the message in corpus list
corpus.append(message)
```

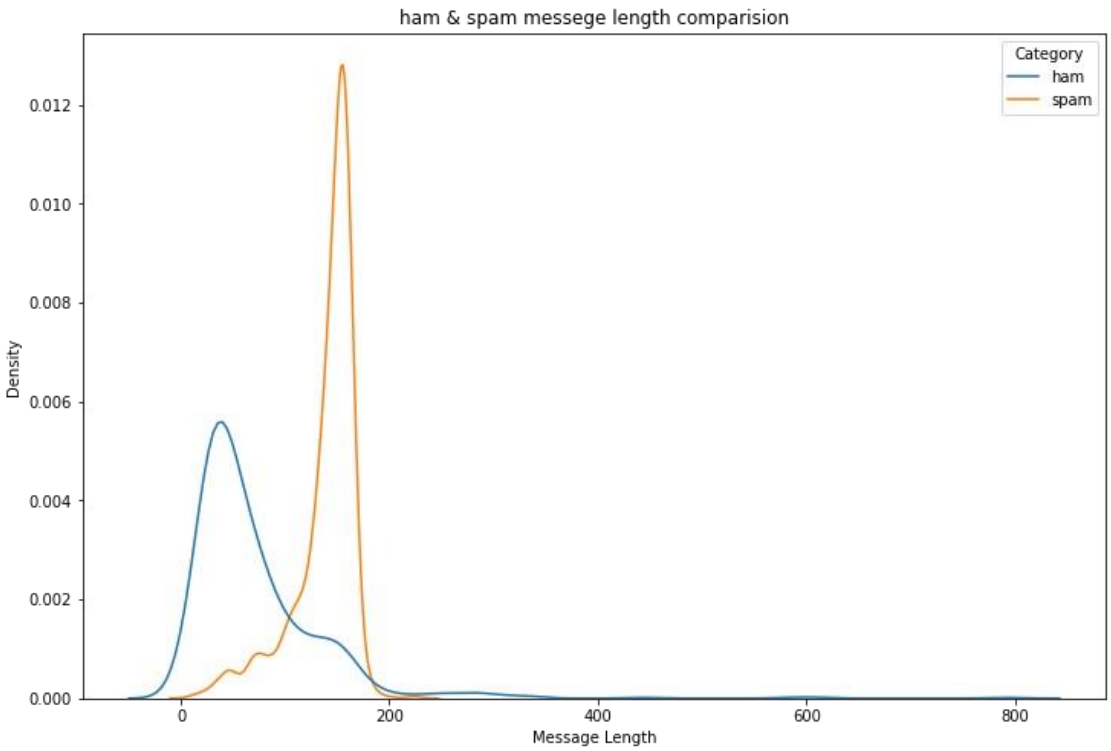
```
from tensorflow.keras.preprocessing.text import one_hot
vocab_size=10000

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

```
df["Message Length"].describe()
```

```
count    1494.000000
mean      105.203481
std        61.166448
min         3.000000
25%        48.000000
50%       118.000000
75%       153.000000
max       790.000000
Name: Message Length, dtype: float64
```

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



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

extract_features=pd.DataFrame(
data=embedded_doc
)
target=df["Label"]

df_final=pd.concat([extract_features,target],axis=1)

df_final.head()
```

```
   0  1  2  3  4  5  6  7  8  9  ...   191   192   193   194   195
196  \
0  0  0  0  0  0  0  0  0  0  0  ...  2090  1632  4289  7158   478
   5808
1  0  0  0  0  0  0  0  0  0  0  ...     0     0     0     0     0
   0
2  0  0  0  0  0  0  0  0  0  0  ...  1275   702  1694  4114  4162
3935
3  0  0  0  0  0  0  0  0  0  0  ...  3705  9946  5462  7158  9883
4500
4  0  0  0  0  0  0  0  0  0  0  ...  4753  6414  5018  1953   216
   1175
```

```
    197   198   199  Label
0  6133  8348  4198      0
1  8663  4425  6636      0
2  4162  8536  7201      1
3  8030  8630  2977      0
4  8861  2485  6055      1

[5 rows x 201 columns]
```

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

from sklearn.model_selection import train_test_split

X_trainval,X_test,y_trainval,y_test=train_test_split(
X,     y,     random_state=42,     test_size=0.15 )

X_train,X_val,y_train,y_val=train_test_split(
X_trainval,     y_trainval,
random_state=42,     test_size=0.15
)

from tensorflow.keras.layers import LSTM from
tensorflow.keras.layers import Dense from
tensorflow.keras.layers import Embedding from
tensorflow.keras.models import Sequential
model=Sequential()

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"
    )
)
```

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

```
model.fit(
X_train,     y_train,
validation_data=(
X_val,         y_val
),     epochs=10 )
```

```
Epoch 1/10
34/34 [==============================] - 8s 33ms/step - loss: 0.5258 -
accuracy: 0.7653 - val_loss: 0.3215 - val_accuracy: 0.8691
Epoch 2/10
34/34 [==============================] - 1s 16ms/step - loss: 0.1718 -
accuracy: 0.9453 - val_loss: 0.1003 - val_accuracy: 0.9738
Epoch 3/10
34/34 [==============================] - 1s 16ms/step - loss: 0.0533 -
accuracy: 0.9842 - val_loss: 0.0764 - val_accuracy: 0.9791
Epoch 4/10
34/34 [==============================] - 1s 15ms/step - loss: 0.0254 -
accuracy: 0.9926 - val_loss: 0.0716 - val_accuracy: 0.9843
Epoch 5/10
34/34 [==============================] - 1s 16ms/step - loss: 0.0184 -
accuracy: 0.9954 - val_loss: 0.0728 - val_accuracy: 0.9843
Epoch 6/10
34/34 [==============================] - 1s 16ms/step - loss: 0.0134 -
accuracy: 0.9963 - val_loss: 0.0852 - val_accuracy: 0.9843
Epoch 7/10
34/34 [==============================] - 1s 16ms/step - loss: 0.0150 -
accuracy: 0.9954 - val_loss: 0.0744 - val_accuracy: 0.9791
Epoch 8/10
34/34 [==============================] - 1s 16ms/step - loss: 0.0112 -
accuracy: 0.9972 - val_loss: 0.0657 - val_accuracy: 0.9843
Epoch 9/10
34/34 [==============================] - 1s 16ms/step - loss: 0.0062
accuracy: 0.9981 - val_loss: 0.0732 - val_accuracy: 0.9843
Epoch 10/10
34/34 [==============================] - 1s 16ms/step - loss: 0.0050
accuracy: 0.9991 - val_loss: 0.0843 - val_accuracy: 0.9843
```

```
<keras.callbacks.History at 0x7fa3263a7850>
```

```
y_pred=model.predict(X_test)
y_pred=(y_pred>0.5)
```

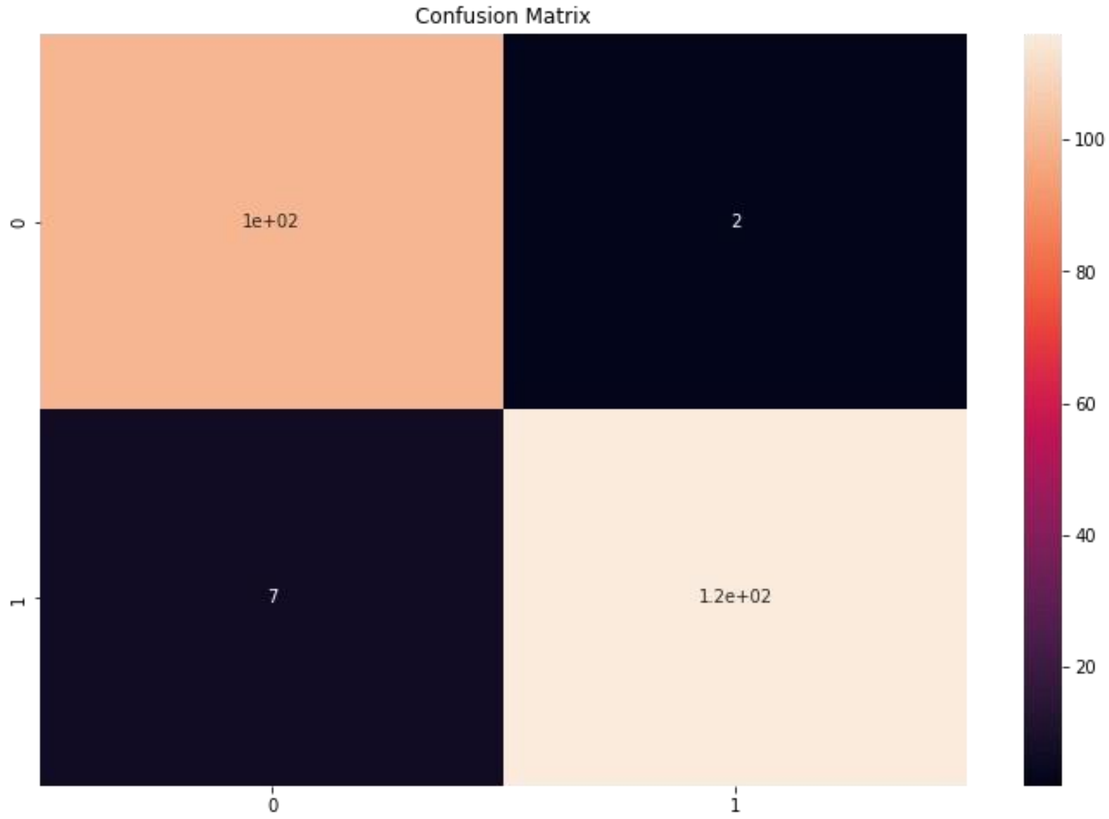
```
8/8 [==============================] - 0s 8ms/step from

sklearn.metrics import accuracy_score,confusion_matrix

score=accuracy_score(y_test,y_pred)
print("Test Score:{:.2f}%".format(score*100))
Test Score:96.00%

cm=confusion_matrix(y_test,y_pred)
fig=plt.figure(figsize=(12,8))
sns.heatmap(      cm,
annot=True,
)
plt.title("Confusion Matrix")
cm

array([[100,   2],
[  7, 116]])
```



```
#The function take model and message as parameter
def classify_message(model,message):

    #We will treat message as a paragraphs containing multiple
sentences(lines)
```

```
    #we will extract individual lines     for
sentences in message:
sentences=nltk.sent_tokenize(message)

        #Iterate over individual sentences
for sentence in sentences:             #replace
all special characters
words=re.sub("[^a-zA-Z]"," ",sentence)

            #perform word tokenization of all non-english-stopwords
if words not in set(stopwords.words('english')):
word=nltk.word_tokenize(words)
                word=" ".join(word)

    #perform one_hot on tokenized word
oneHot=[one_hot(word,n=vocab_size)]

    #create an embedded documnet using pad_sequences
    #this can be fed to our model
    text=pad_sequences(oneHot,maxlen=sentence_len,padding="pre")
#predict the text using model     predict=model.predict(text)

    #if predict value is greater than 0.5 its a spam
if predict>0.5:         print("It is a spam")
    #else the message is not a spam
else:         print("It is not a
spam")

message1="I am having a bad day and I would like to have a break
today"
message2="This is to inform you had won a lottery and the subscription
will end in a week so call us." nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip. True
```

```
classify_message(model,message1)
```

```
1/1 [==============================] - 0s 21ms/step
It is not a spam
```

```
classify_message(model,message2)
```

```
1/1 [==============================] - 0s 22ms/step
It is a spam
```