Assignment -4

Python Programming

Assignment Date	21 October 2022
Student Name	GOPINATH.S
Student Roll Number	311419205010
Maximum Marks	2 Marks

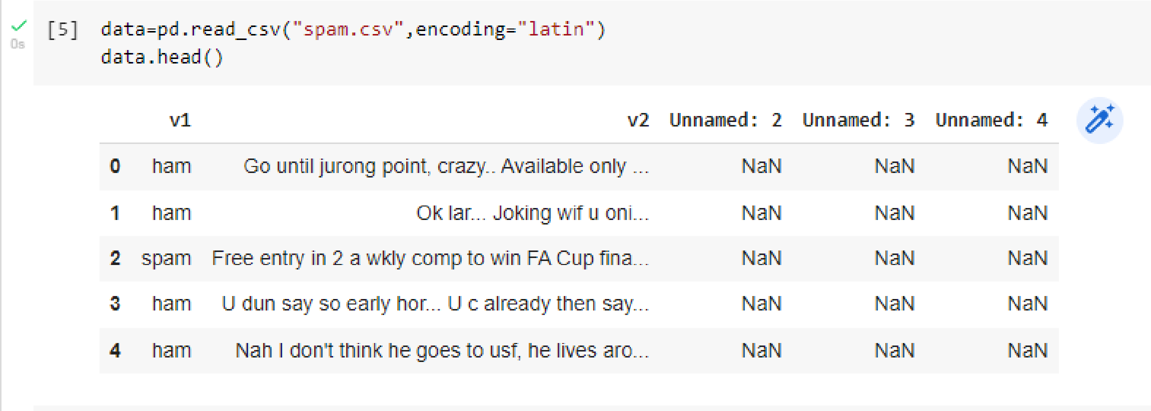
Problem Statement :- SMS SPAM Classification

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

- Download the Dataset:- Dataset
- Import required library
- Read dataset and do pre-processing
- Create Model
- Add Layers (LSTM, Dense-(Hidden Layers), Output)
- Compile the Model
- Fit the Model
- Save The Model
- Test The Mode

Solution:

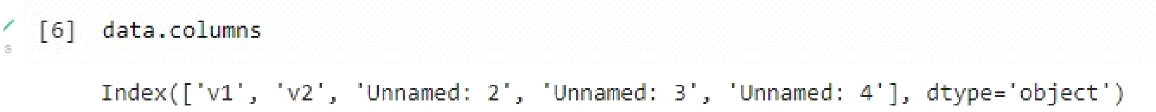
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Embedding

from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score,confusion_matrix
data=pd.read_csv("spam.csv",encoding="latin")
data.head()
```



```
[5] data=pd.read_csv("spam.csv",encoding="latin")
    data.head()
```

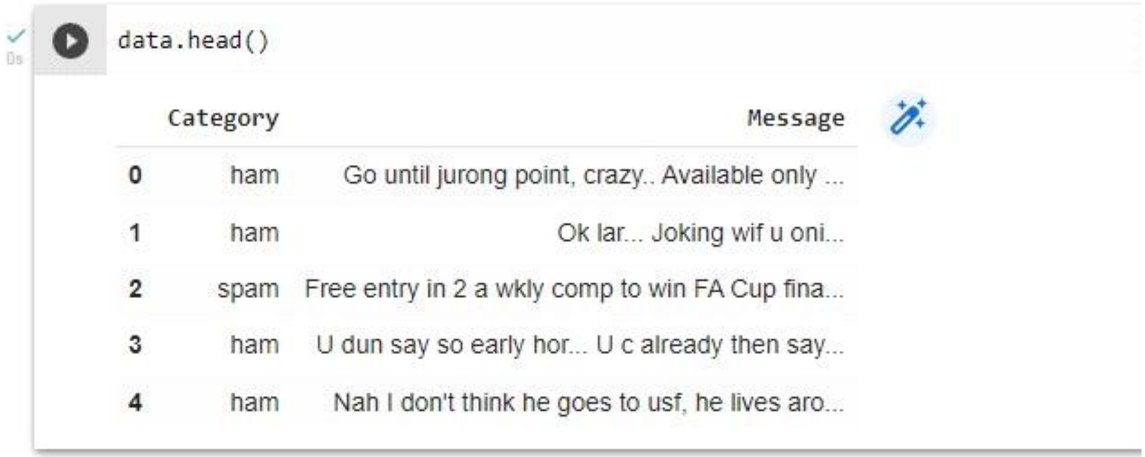
	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah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

```
data.columns
```



```
[6] data.columns

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

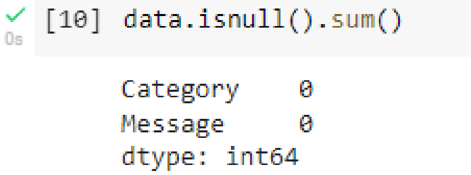
```
data=data.drop(columns=["Unnamed: 2","Unnamed: 3","Unnamed: 4"])
data=data.rename(
{
  "v1":"Category",
  "v2":"Message"
},
  axis=1
)
data.head()
```



```
data.head()
```

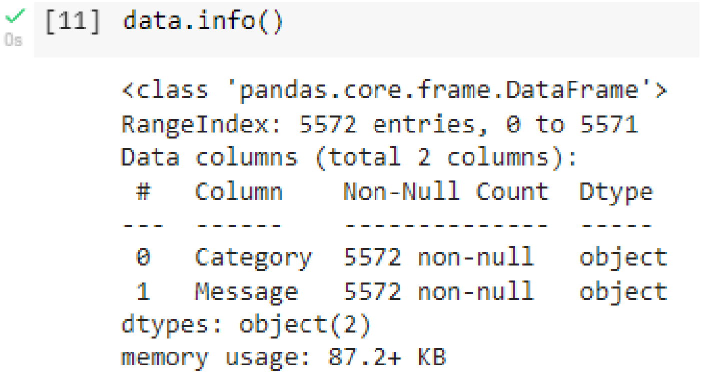
	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

data.isnull().sum()



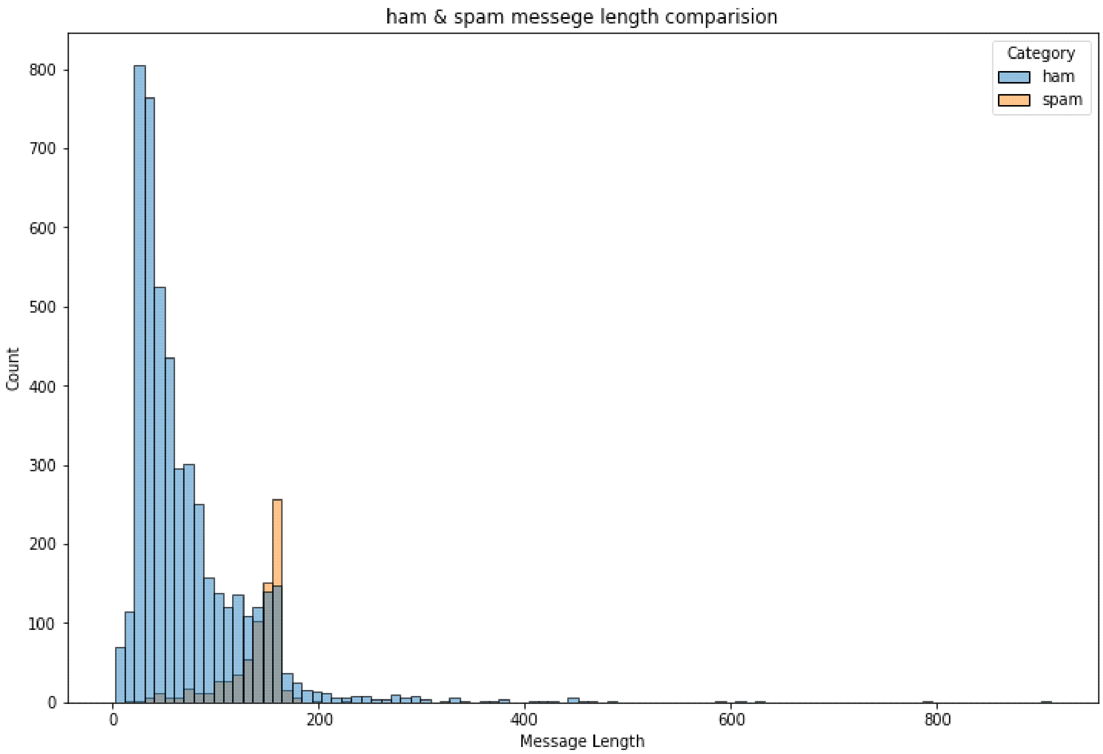
```
[10] data.isnull().sum()

     Category    0
     Message     0
     dtype: int64
```

data.info()

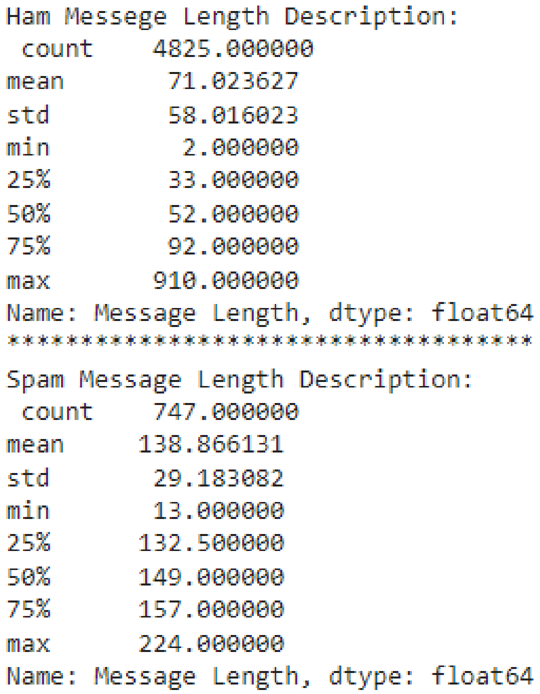


```
[11] data.info()

     <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()
```

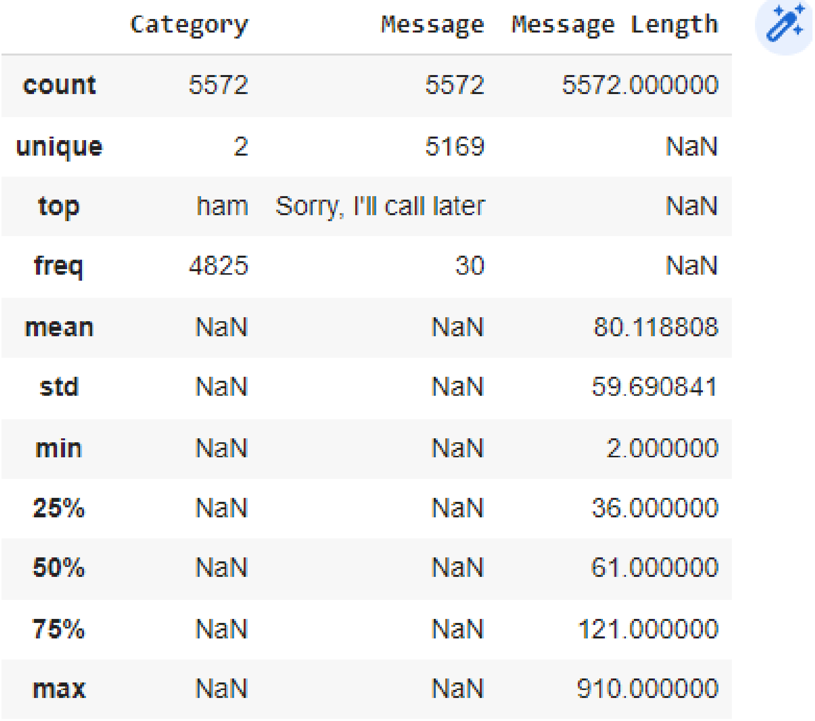


```
ham_desc=data[data["Category"]=="ham"]["Message Length"].describe()
spam_desc=data[data["Category"]=="spam"]["Message Length"].describe()

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

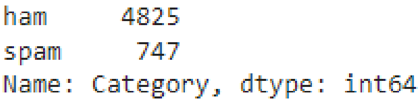


```
Ham Messege Length Description:
 count    4825.000000
mean       71.023627
std        58.016023
min         2.000000
25%        33.000000
50%        52.000000
75%        92.000000
max       910.000000
Name: Message Length, dtype: float64
************************************
Spam Message Length Description:
 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")



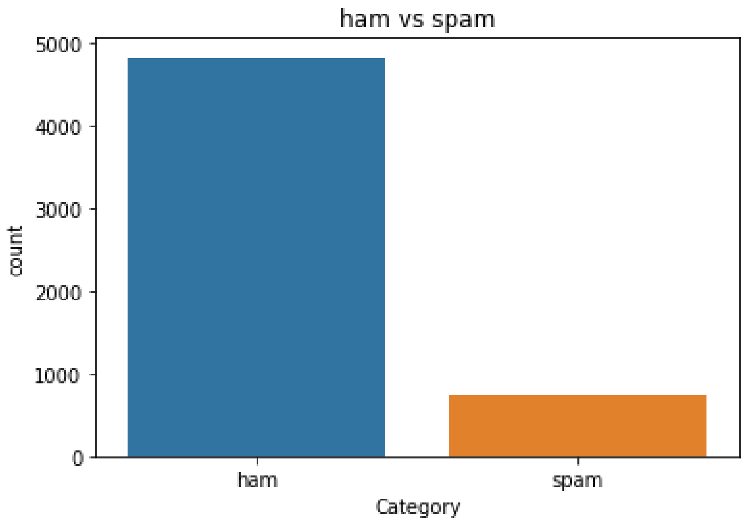
	Category	Message	Message Length
count	5572	5572	5572.000000
unique	2	5169	NaN
top	ham	Sorry, I'll call later	NaN
freq	4825	30	NaN
mean	NaN	NaN	80.118808
std	NaN	NaN	59.690841
min	NaN	NaN	2.000000
25%	NaN	NaN	36.000000
50%	NaN	NaN	61.000000
75%	NaN	NaN	121.000000
max	NaN	NaN	910.000000



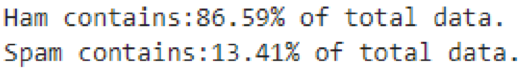
```
ham     4825
spam     747
Name: Category, dtype: int64
```

data["Category"].value_counts()

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



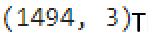
```
ham_count=data["Category"].value_counts()[0]
spam_count=data["Category"].value_counts()[1]

total_count=data.shape[0]
```



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

```
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))

#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])
```

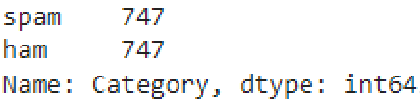
```
#create df using new indices
df=data.loc[undersampled_indices]


#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"],
)
```



```
(1494, 3)
```

The resulting dataframes have **1494** rows and

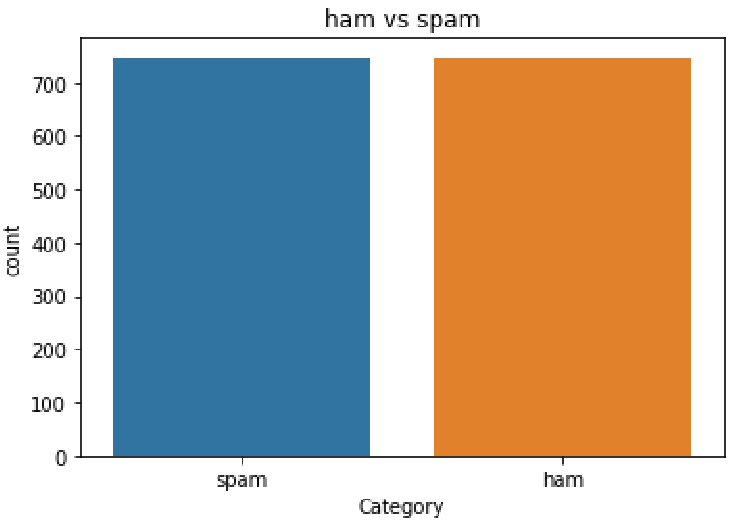
4 columnsdf.shape



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

df["Category"].value_counts()

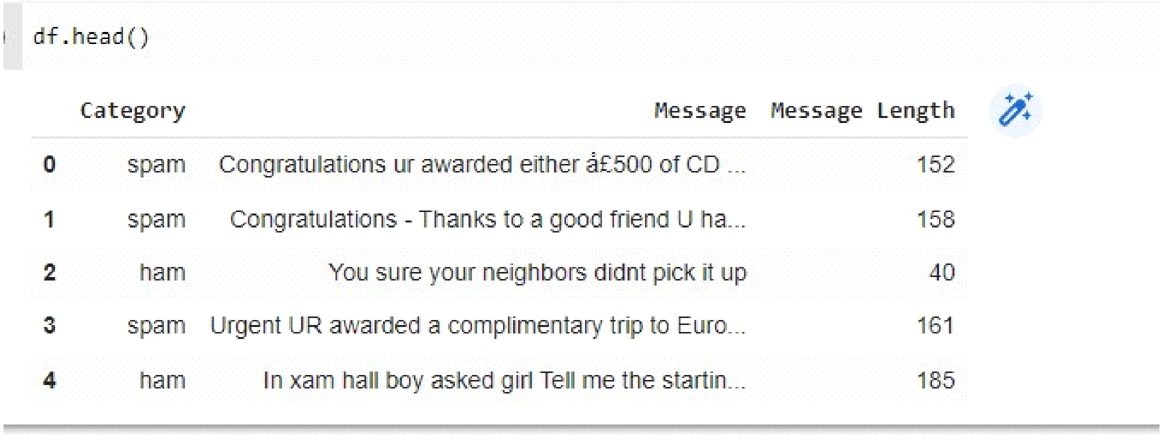
```
sns.countplot(
   data=df,
   x="Category"
)
```



```
plt.title("ham vs spam")
plt.show()
```

Display the head of new **df**

```
df.head()
```



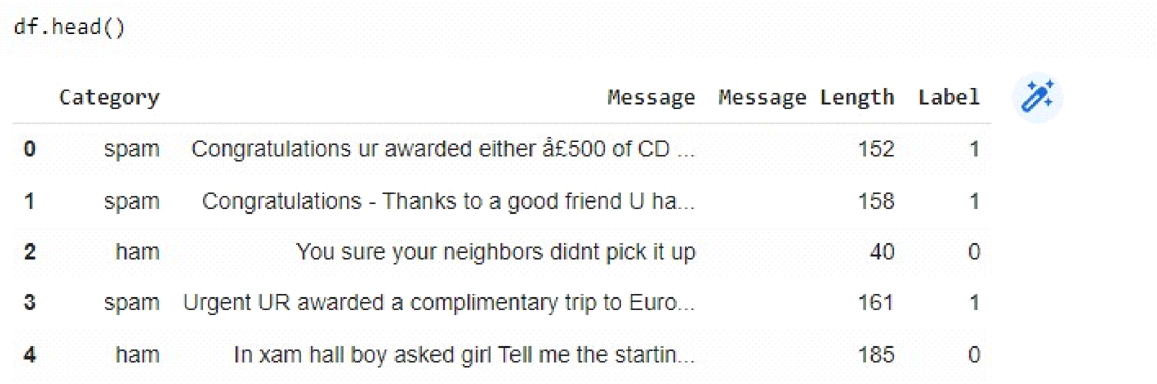
```
df.head()
```

```
   Category                                      Message  Message Length
0      spam  Congratulations ur awarded either å£500 of CD ...           152
1      spam   Congratulations - Thanks to a good friend U ha...           158
2       ham              You sure your neighbors didnt pick it up            40
3      spam  Urgent UR awarded a complimentary trip to Euro...           161
4       ham         In xam hall boy asked girl Tell me the startin...           185
```

Created new column **Label** and encode **ham** as **0** and **spam** as **1**

```
df["Label"]=df["Category"].map(

   {
      "ham":0,
      "spam":1
   }
)
```

```
df.head()
```



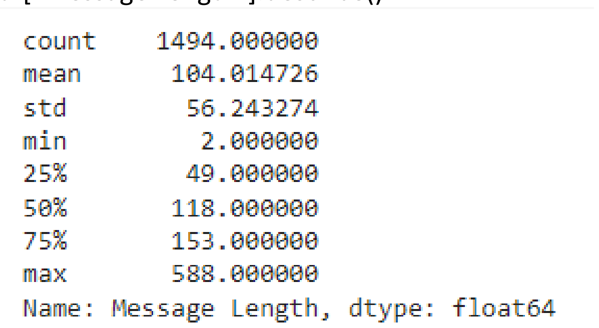
```
df.head()
```

	Category	Message	Message Length	Label
0	spam	Congratulations ur awarded either å£500 of CD ...	152	1
1	spam	Congratulations - Thanks to a good friend U ha...	158	1
2	ham	You sure your neighbors didnt pick it up	40	0
3	spam	Urgent UR awarded a complimentary trip to Euro...	161	1
4	ham	In xam hall boy asked girl Tell me the startin...	185	0

Import libraries to perform word **tokenization**

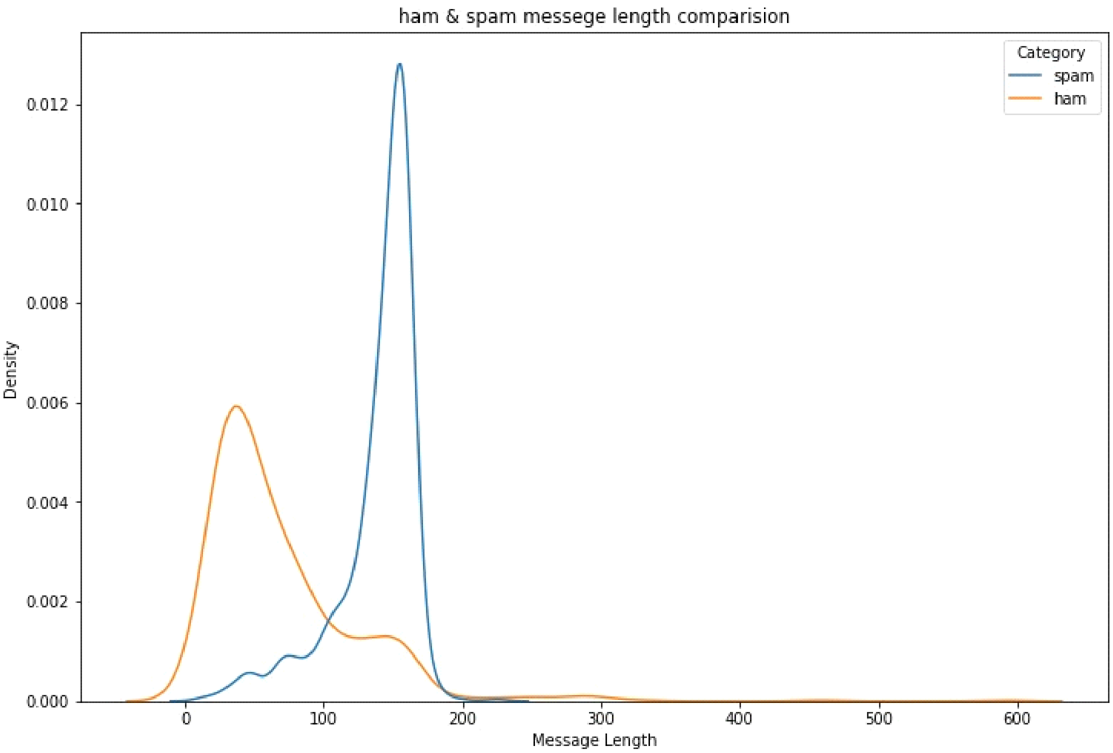
```
stemmer=PorterStemmer()

#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-english-stopwords
   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)

vocab_size=10000


oneHot_doc=[one_hot(words,n=vocab_size)


          for words in corpus
          ]

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

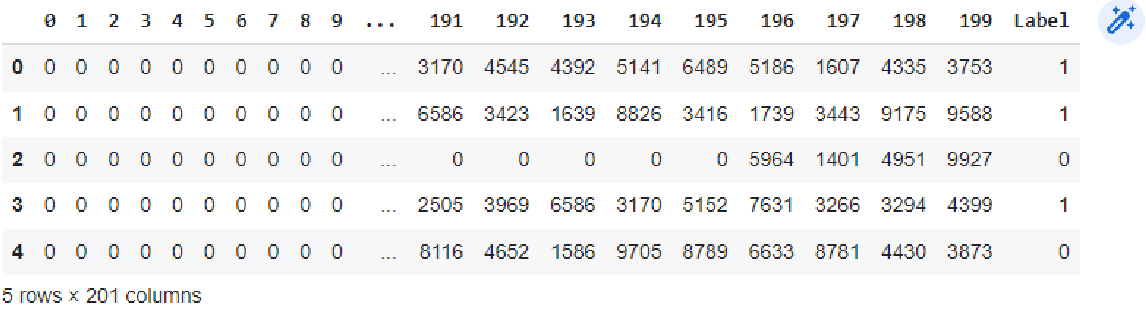


```
count    1494.000000
mean      104.014726
std        56.243274
min         2.000000
25%        49.000000
50%       118.000000
75%       153.000000
max       588.000000
Name: Message Length, dtype: float64
```

```
fig=plt.figure(figsize=(12,8))
sns.kdeplot(
    x=df["Message Length"],
    hue=df["Category"]
)

plt.title("ham & spam messege length comparision")
plt.show()
```



```
sentence_len=200
embedded_doc=pad_sequences(
   oneHot_doc,
   maxlen=sentence_len,
   padding="pre"
)

extract_features=pd.DataFrame(
   data=embedded_doc
)
```



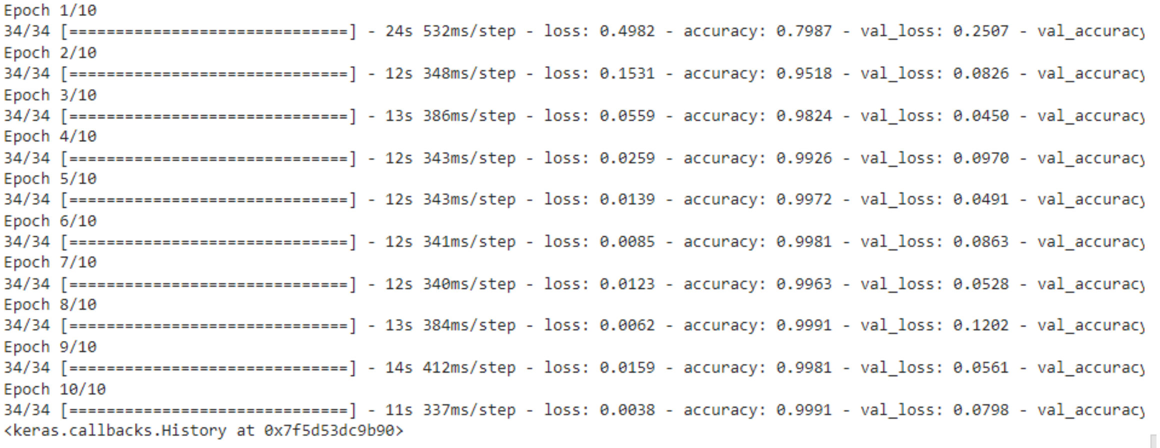
	0	1	2	3	4	5	6	7	8	9	...	191	192	193	194	195	196	197	198	199	Label
0	0	0	0	0	0	0	0	0	0	0	...	3170	4545	4392	5141	6489	5186	1607	4335	3753	1
1	0	0	0	0	0	0	0	0	0	0	...	6586	3423	1639	8826	3416	1739	3443	9175	9588	1
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	5964	1401	4951	9927	0
3	0	0	0	0	0	0	0	0	0	0	...	2505	3969	6586	3170	5152	7631	3266	3294	4399	1
4	0	0	0	0	0	0	0	0	0	0	...	8116	4652	1586	9705	8789	6633	8781	4430	3873	0

5 rows × 201 columns

```
target=df["Label"]

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

df_final.head()
```

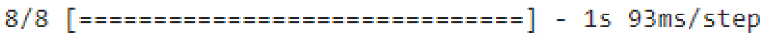
```
X=df_final.drop("Label",axis=1)
y=df_final["Label"]

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
)

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"
    )
)
```

```
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
)
```

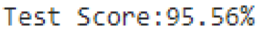


```
Epoch 1/10
34/34 [==============================] - 24s 532ms/step - loss: 0.4982 - accuracy: 0.7987 - val_loss: 0.2507 - val_accuracy
Epoch 2/10
34/34 [==============================] - 12s 348ms/step - loss: 0.1531 - accuracy: 0.9518 - val_loss: 0.0826 - val_accuracy
Epoch 3/10
34/34 [==============================] - 13s 386ms/step - loss: 0.0559 - accuracy: 0.9824 - val_loss: 0.0450 - val_accuracy
Epoch 4/10
34/34 [==============================] - 12s 343ms/step - loss: 0.0259 - accuracy: 0.9926 - val_loss: 0.0970 - val_accuracy
Epoch 5/10
34/34 [==============================] - 12s 343ms/step - loss: 0.0139 - accuracy: 0.9972 - val_loss: 0.0491 - val_accuracy
Epoch 6/10
34/34 [==============================] - 12s 341ms/step - loss: 0.0085 - accuracy: 0.9981 - val_loss: 0.0863 - val_accuracy
Epoch 7/10
34/34 [==============================] - 12s 340ms/step - loss: 0.0123 - accuracy: 0.9963 - val_loss: 0.0528 - val_accuracy
Epoch 8/10
34/34 [==============================] - 13s 384ms/step - loss: 0.0062 - accuracy: 0.9991 - val_loss: 0.1202 - val_accuracy
Epoch 9/10
34/34 [==============================] - 14s 412ms/step - loss: 0.0159 - accuracy: 0.9981 - val_loss: 0.0561 - val_accuracy
Epoch 10/10
34/34 [==============================] - 11s 337ms/step - loss: 0.0038 - accuracy: 0.9991 - val_loss: 0.0798 - val_accuracy
<keras.callbacks.History at 0x7f5d53dc9b90>
```

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



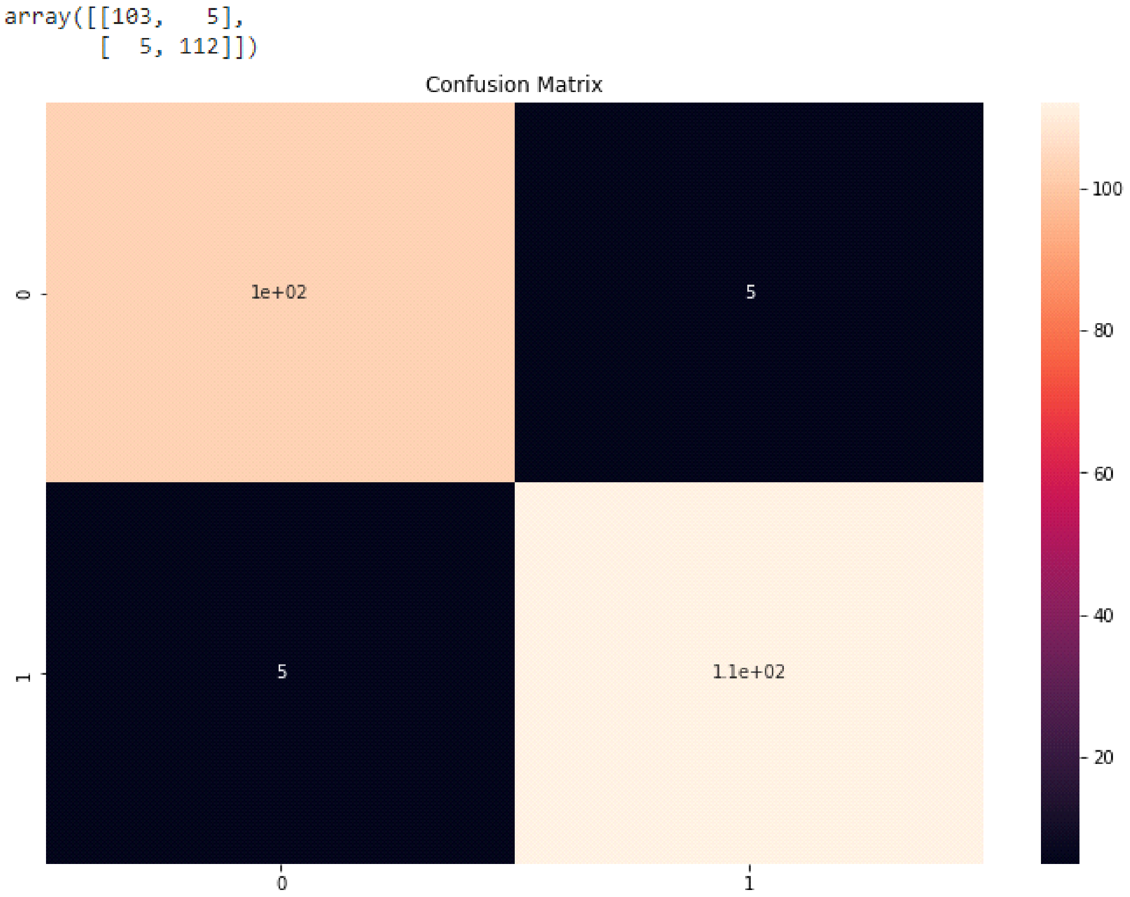
```
8/8 [==============================] - 1s 93ms/step
```



```
Test Score:95.56%
```

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

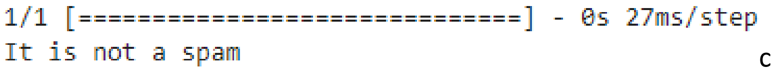
```
cm=confusion_matrix(y_test,y_pred)
fig=plt.figure(figsize=(12,8))
sns.heatmap(
    cm,
    annot=True,
)
plt.title("Confusion Matrix")
cm
```



```
array([[103,   5],
       [  5, 112]])
```

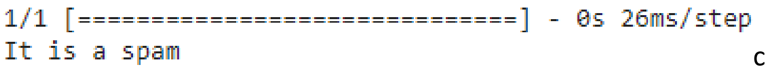
```
#The function take model and message as parameter
def classify_message(model,message):

    #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."
```



```
1/1 [==============================] - 0s 27ms/step
It is not a spam
```

classify_message(model,message1)



```
1/1 [==============================] - 0s 26ms/step
It is a spam
```

classify_message(model,message2)