

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input,
Embedding
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
%matplotlib inline

from tensorflow.keras.preprocessing.sequence import pad_sequences
df = pd.read_csv('spam.csv', delimiter=',', encoding='latin-1')

df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1, inplace=True)
df.info()

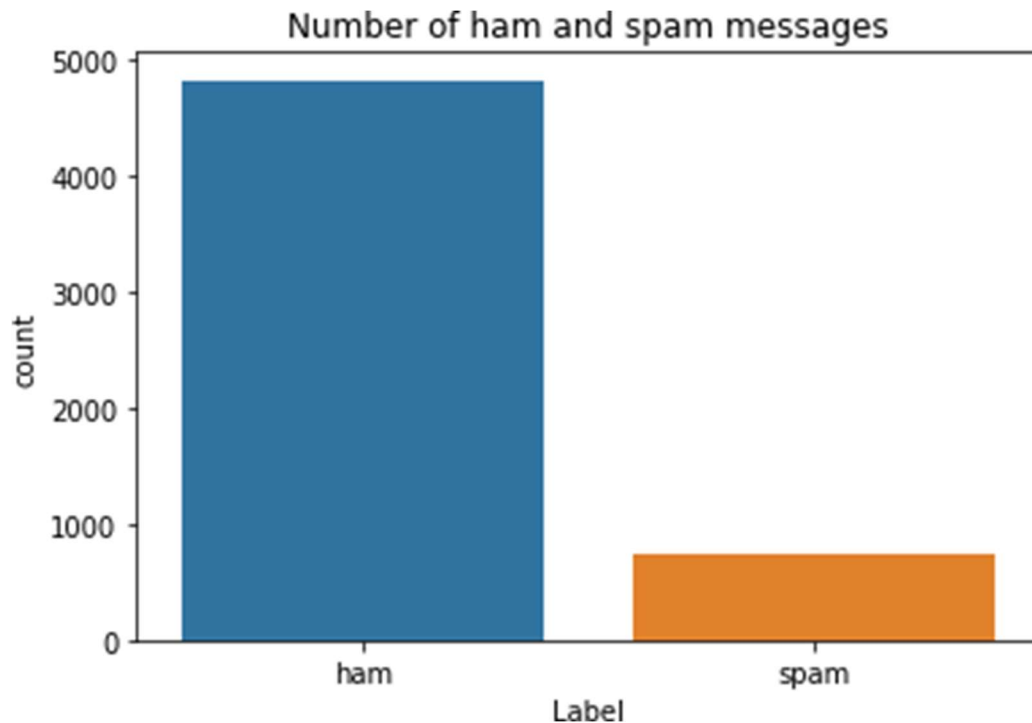
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571 Data
columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  -
0    v1      5572 non-null     object
1    v2      5572 non-null     object
dtypes: object(2)
memory usage: 87.2+ KB

sns.countplot(df.v1)
plt.xlabel('Label')
plt.title('Number of ham and spam messages')

/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
FutureWarning

Text(0.5, 1.0, 'Number of ham and spam messages')

```



```

X = df.v2
Y = df.v1
le = LabelEncoder()
Y = le.fit_transform(Y) Y =
Y.reshape(-1,1)

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.15) max_words
= 1000
max_len = 150
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(X_train)
sequences = tok.texts_to_sequences(X_train)

def RNN():
    inputs = Input(name='inputs',shape=[max_len])
    layer = Embedding(max_words,50,input_length=max_len)(inputs)
    layer = LSTM(64)(layer)
    layer = Dense(256,name='FC1')(layer)
    layer = Activation('relu')(layer) layer =
    Dropout(0.5)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model

model = RNN()
model.summary()

```

```
model.compile(loss='binary_crossentropy',optimizer=RMSprop(),metrics=[
'accuracy'])
```

Model: "model"

Layer (type)	Output Shape	Param #
inputs (InputLayer)	[(None, 150)]	0
embedding (Embedding)	(None, 150, 50)	50000
lstm (LSTM)	(None, 64)	29440
FC1 (Dense)	(None, 256)	16640
activation (Activation)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
out_layer (Dense)	(None, 1)	257
activation_1 (Activation)	(None, 1)	0
Total params: 96,337		
Trainable params: 96,337		
Non-trainable params: 0		

```
df.columns
```

```
Index(['v1', 'v2', 'Count'], dtype='object')
```

```
data=df.rename(
{
    "v1":"Category",
    "v2":"Message"
},
axis=1
)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5572 entries, 1211 to 3623 Data
columns (total 3 columns):
```

#	Column	Non-Null Count	Dtype
0	v1	5572 non-null	object
1	v2	5572 non-null	object
2	Count	5572 non-null	int64

```
dtypes: int64(1), object(2)
memory usage: 174.1+ KB
```

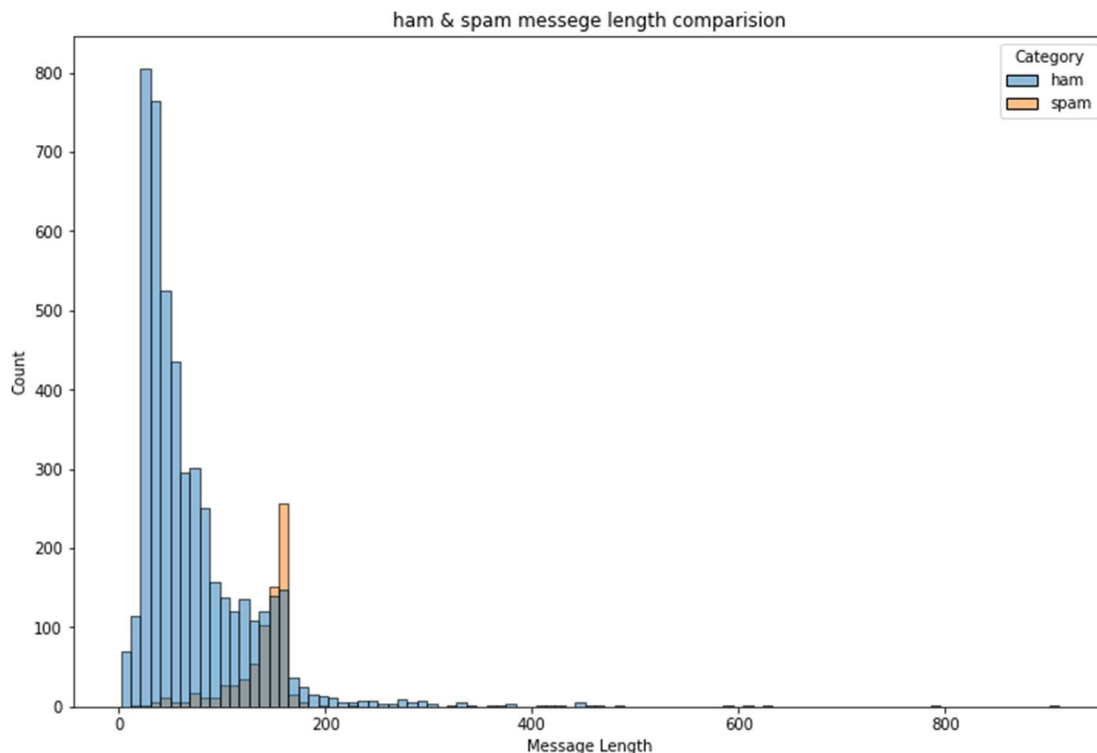
```
data["Message Length"]=data["Message"].apply(len)
```

```
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
```

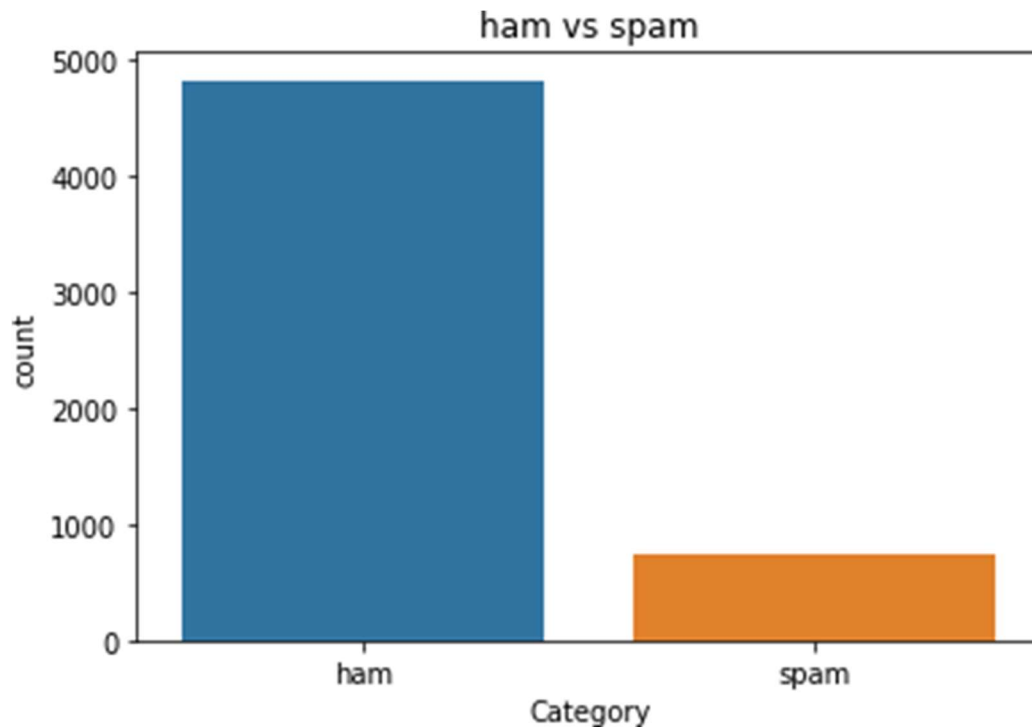
```
data.describe(include="all")
```

	Category	Message	Count	Message Length
count	5572	5572	5572.0	5572.000000
unique	2	5169	NaN	NaN
top	ham Sorry, I'll	call later	NaN	NaN
freq	4825	30	NaN	NaN
mean	NaN	NaN	0.0	80.118808
std	NaN	NaN	0.0	59.690841
min	NaN	NaN	0.0	2.000000
25%	NaN	NaN	0.0	36.000000
50%	NaN	NaN	0.0	61.000000
75%	NaN	NaN	0.0	121.000000
max	NaN	NaN	0.0	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

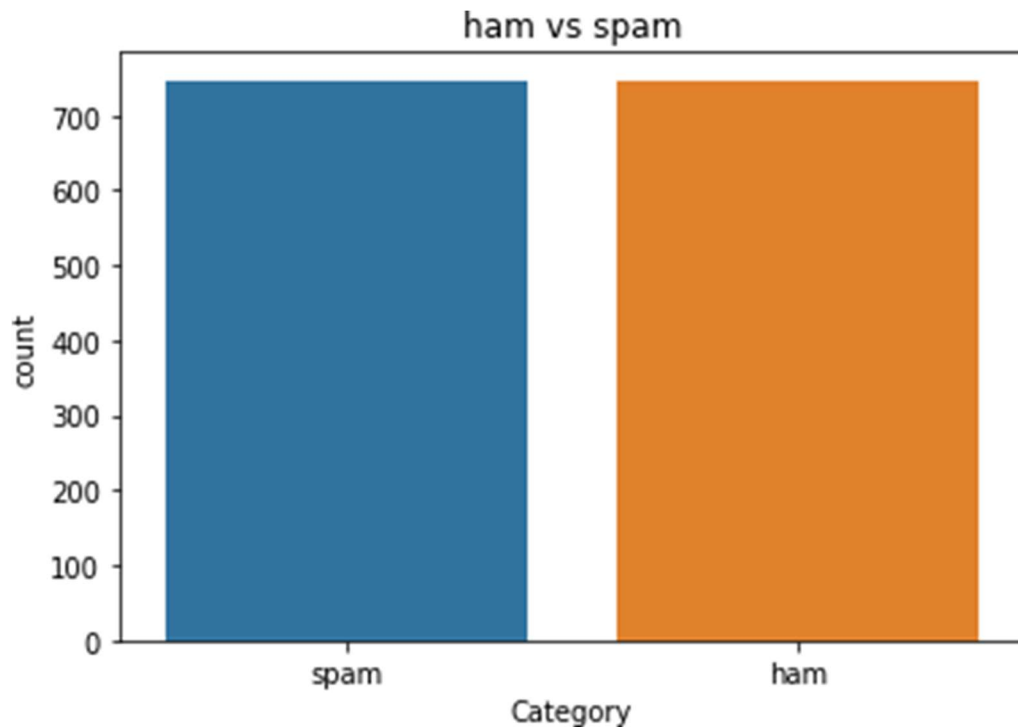
    (1494, 4)

    df["Category"].value_counts()

    spam    747
    ham      747
    Name: Category, dtype: int64

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

```



```
df.head()
```

	Category	Message	
Count \			
0	spam	Eerie Nokia tones 4u, rply TONE TITLE to 8007 ...	0
1	ham	That sucks. I'll go over so u can do my hair. ...	0
2	ham	says that he's quitting at least5times a day ...	0
3	ham	Hey. For me there is no leave on friday. Wait ...	0
4	spam	Please call our customer service representativ...	0

	Message Length
0	162
1	70
2	200
3	83
4	149

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



```
    }
)
```

```
df.head()
```

	Category	Message	
Count \			
0	spam	Eerie Nokia tones 4u, rply TONE TITLE to 8007 ...	0
1	ham	That sucks. I'll go over so u can do my hair. ...	0
2	ham	says that he's quitting at least5times a day ...	0
3	ham	Hey. For me there is no leave on friday. Wait ...	0
4	spam	Please call our customer service representativ...	0

	Message Length	Label
0	162	1
1	70	0
2	200	0
3	83	0
4	149	1

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

```
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()
```

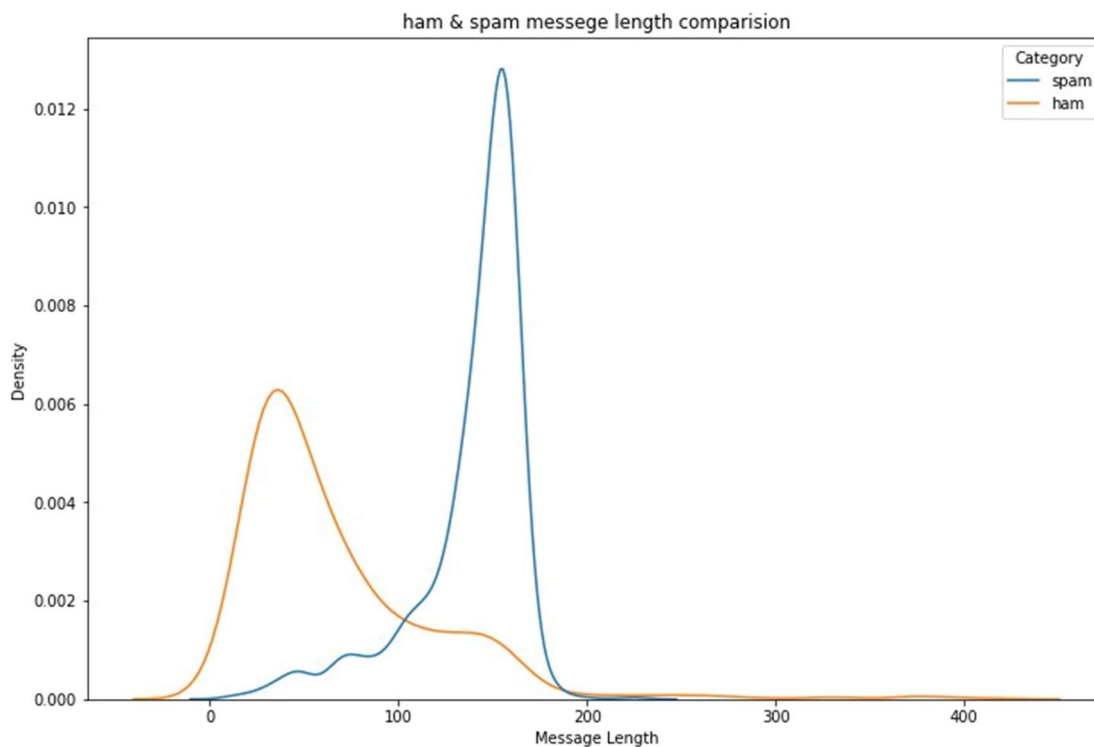
```
from tensorflow.keras.preprocessing.text import one_hot
vocab_size=10000
```

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

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

```
count      1494.000000
mean       103.384873
std        55.635473
min         2.000000
25%        48.000000
50%       115.000000
75%       152.750000
max       408.000000
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	NaN	NaN		Na	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	Na	Na
196	NaN	NaN		N										N	N
0	NaN	NaN		Na	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	Na	Na
1	NaN	NaN		N										N	N
2	NaN	NaN		Na	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	Na	Na
3	NaN	NaN		N										N	N
4	NaN	NaN		Na	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	Na	Na
	NaN														
197	NaN	NaN													
198	NaN	NaN													
199	NaN	NaN													
Label															

	197	198	199	Label
0	NaN	NaN	NaN	1
1	NaN	NaN	NaN	0
2	NaN	NaN	NaN	0
3	NaN	NaN	NaN	0
4	NaN	NaN	NaN	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
)

model = RNN()
model.summary()
model.compile(loss='binary_crossentropy',optimizer=RMSprop(),metrics=[
'accuracy'])

```

Model: "model\_3"

Layer (type)	Output Shape	Param #
=====		
inputs (InputLayer)	[(None, 150)]	0
embedding_4 (Embedding)	(None, 150, 50)	50000
lstm_4 (LSTM)	(None, 64)	29440
FC1 (Dense)	(None, 256)	16640
activation_6 (Activation)	(None, 256)	0
dropout_3 (Dropout)	(None, 256)	0
out_layer (Dense)	(None, 1)	257
activation_7 (Activation)	(None, 1)	0
=====		
Total params: 96,337		
Trainable params: 96,337		
Non-trainable params: 0		