```
#SMS-SPAM Classification

import os
import re
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, Embedding
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.text import Tokenizer
import keras
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

#Reading The File

df = pd.read_csv(filepath_or_buffer='spam.csv',
delimiter=',',encoding='latin-1')
df.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  

#Column Names
df.columns
```

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

```
#Removing Unnamed Columns
df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1,
inplace=True)
df.columns
```

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

```
#Number of Rows
df.shape
```

```
(5572, 2)
```

```
#Summary of dataset
df.describe()
```

```
          v1                        v2
count   5572                      5572
unique     2                      5169
top      ham  Sorry, I'll call later
freq    4825                        30
```

```
#Checking for null values
df.isna().sum()
```

```
v1    0
v2    0
dtype: int64
```

```
#Checking for duplicate rows
df.duplicated().sum()
```

```
403
```

```
#Removing the duplicate rows
df = df.drop_duplicates()
df.duplicated().sum()
```

```
0
```

```
#Text Processing
df['alpha'] = df['v2'].apply(lambda x: re.sub(r'[^a-zA-Z ]+',
'',x.lower()))
df.head()
```

```
     v1                                                  v2  \
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...
```

```
                                               alpha
0  go until jurong point crazy available only in ...
1                            ok lar joking wif u oni
2  free entry in  a wkly comp to win fa cup final...
3        u dun say so early hor u c already then say
4  nah i dont think he goes to usf he lives aroun...
```

```
#Removing stop-words
nltk.download('stopwords')
df['imp_text'] = df['alpha'].apply(lambda x : ' '.join([word for word
in x.split() if not word in set(stopwords.words('english'))]))
df.head()
```

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

```
     v1                                                 v2  \
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...

                                               alpha  \
0  go until jurong point crazy available only in ...
1                            ok lar joking wif u oni
2  free entry in  a wkly comp to win fa cup final...
3        u dun say so early hor u c already then say
4  nah i dont think he goes to usf he lives aroun...

                                            imp_text
0  go jurong point crazy available bugis n great ...
1                            ok lar joking wif u oni
2  free entry wkly comp win fa cup final tkts st ...
3                  u dun say early hor u c already say
4        nah dont think goes usf lives around though
```

```
#Tokenize the data
def tokenize(data):
  generated_token = list(data.split())
  return generated_token
df['token_text'] = df['imp_text'].apply(lambda x: tokenize(x))
df.head()
```

```
     v1                                                 v2  \
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...
```

```
                                                      alpha  \
0  go until jurong point crazy available only in ...
1                             ok lar joking wif u oni
2  free entry in  a wkly comp to win fa cup final...
3        u dun say so early hor u c already then say
4  nah i dont think he goes to usf he lives aroun...

                                                   imp_text  \
0  go jurong point crazy available bugis n great ...
1                             ok lar joking wif u oni
2  free entry wkly comp win fa cup final tkts st ...
3                u dun say early hor u c already say
4        nah dont think goes usf lives around though

                                                 token_text
0  [go, jurong, point, crazy, available, bugis, n...
1                      [ok, lar, joking, wif, u, oni]
2  [free, entry, wkly, comp, win, fa, cup, final,...
3      [u, dun, say, early, hor, u, c, already, say]
4  [nah, dont, think, goes, usf, lives, around, t...
```

```
#Perform lemmatization
nltk.download('wordnet')
nltk.download('omw-1.4')
lemmatizer = WordNetLemmatizer()
def lemmatization(list_of_words):
  lemmatized_list = [lemmatizer.lemmatize(word) for word in
list_of_words]
  return lemmatized_list
df['lemmatized_text'] = df['token_text'].apply(lambda x:
lemmatization(x))
df.head()
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...

     v1                                                 v2  \
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...

                                                      alpha  \
0  go until jurong point crazy available only in ...
1                             ok lar joking wif u oni
2  free entry in  a wkly comp to win fa cup final...
3        u dun say so early hor u c already then say
4  nah i dont think he goes to usf he lives aroun...
```

```
                                            imp_text  \
0  go jurong point crazy available bugis n great ...
1                          ok lar joking wif u oni
2  free entry wkly comp win fa cup final tkts st ...
3                u dun say early hor u c already say
4        nah dont think goes usf lives around though

                                          token_text  \
0  [go, jurong, point, crazy, available, bugis, n...
1                   [ok, lar, joking, wif, u, oni]
2  [free, entry, wkly, comp, win, fa, cup, final,...
3      [u, dun, say, early, hor, u, c, already, say]
4  [nah, dont, think, goes, usf, lives, around, t...

                                     lemmatized_text
0  [go, jurong, point, crazy, available, bugis, n...
1                   [ok, lar, joking, wif, u, oni]
2  [free, entry, wkly, comp, win, fa, cup, final,...
3      [u, dun, say, early, hor, u, c, already, say]
4  [nah, dont, think, go, usf, life, around, though]
```

```
#Cleaned Dataset
df['clean'] = df['lemmatized_text'].apply(lambda x: ' '.join(x))
df.head()
```

```
     v1                                                 v2  \
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...

                                               alpha  \
0  go until jurong point crazy available only in ...
1                          ok lar joking wif u oni
2  free entry in  a wkly comp to win fa cup final...
3        u dun say so early hor u c already then say
4  nah i dont think he goes to usf he lives aroun...

                                            imp_text  \
0  go jurong point crazy available bugis n great ...
1                          ok lar joking wif u oni
2  free entry wkly comp win fa cup final tkts st ...
3                u dun say early hor u c already say
4        nah dont think goes usf lives around though

                                          token_text  \
0  [go, jurong, point, crazy, available, bugis, n...
1                   [ok, lar, joking, wif, u, oni]
```

```
2  [free, entry, wkly, comp, win, fa, cup, final,...
3      [u, dun, say, early, hor, u, c, already, say]
4  [nah, dont, think, goes, usf, lives, around, t...

                                     lemmatized_text  \
0  [go, jurong, point, crazy, available, bugis, n...
1                        [ok, lar, joking, wif, u, oni]
2  [free, entry, wkly, comp, win, fa, cup, final,...
3      [u, dun, say, early, hor, u, c, already, say]
4  [nah, dont, think, go, usf, life, around, though]

                                               clean
0  go jurong point crazy available bugis n great ...
1                              ok lar joking wif u oni
2  free entry wkly comp win fa cup final tkts st ...
3                 u dun say early hor u c already say
4            nah dont think go usf life around though
```

```
#Number of unique words in spam and ham
df1 = df.loc[df['v1'] == 'spam']
df2 = df.loc[df['v1'] == 'ham']

spam = set()
df1['clean'].str.lower().str.split().apply(spam.update)
print("Number of unique words in spam", len(spam))

ham = set()
df2['clean'].str.lower().str.split().apply(ham.update)
print("Number of unique words in ham", len(ham))
```

```
Number of unique words in spam 2037
Number of unique words in ham 6738
```

```
#Find the number of overlapping words between spam and ham labels
print("Number of overlapping words between spam and ham: ", len(spam & 
ham))
```

```
Number of overlapping words between spam and ham:  895
```

```
#Maximum number of words in a sentence and Useful for applying padding
df['clean'].apply(lambda x:len(str(x).split())).max()
```

```
80
```

```
#Data for training
X = df['clean']
y = df['v1']

#Class Labels -> Integer Values
le = LabelEncoder()
```

```
y = le.fit_transform(y)
y
```

```
array([0, 0, 1, ..., 0, 0, 0])
```

```
X.shape
```

```
(5169,)
```

```
X.shape
```

```
(5169,)
```

```
y.shape
```

```
(5169,)
```

```
#Split the data into train, test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.15, random_state=42, stratify=y)
```

```
tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(X_train)
tokenized_train = tokenizer.texts_to_sequences(X_train)
X_train = tf.keras.utils.pad_sequences(tokenized_train, maxlen=100)
```

```
tokenized_test = tokenizer.texts_to_sequences(X_test)
X_test = tf.keras.utils.pad_sequences(tokenized_test, maxlen=100)
```

```
                                     #Creating The Model
```

```
#Create a wrapper to add layers to the model
model = Sequential()
```

```
#Adding Layers
model.add(Embedding(1000, output_dim=50, input_length=100))
model.add(LSTM(units=64 , return_sequences = True, dropout = 0.2))
model.add(LSTM(units=32 , dropout = 0.1))
model.add(Dense(units = 64 , activation = 'relu'))
model.add(Dense(units = 32 , activation = 'relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 50)           50000

 lstm (LSTM)                 (None, 100, 64)           29440

 lstm_1 (LSTM)               (None, 32)                12416
 dense (Dense)               (None, 64)                2112

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 96,081
Trainable params: 96,081
Non-trainable params: 0
_________________________________________________________________
```

```
#Compiling Model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

```
#Training The Model
model.fit(X_train, y_train,
batch_size=128,epochs=10,validation_split=0.2,callbacks=[EarlyStopping
(monitor='val_loss',patience=2)])
```

```
Epoch 1/10
28/28 [==============================] - 8s 179ms/step - loss: 0.4882
- accuracy: 0.8480 - val_loss: 0.3722 - val_accuracy: 0.8760
Epoch 2/10
28/28 [==============================] - 4s 152ms/step - loss: 0.3472
- accuracy: 0.8736 - val_loss: 0.2317 - val_accuracy: 0.8987
Epoch 3/10
28/28 [==============================] - 4s 154ms/step - loss: 0.1386
- accuracy: 0.9616 - val_loss: 0.0886 - val_accuracy: 0.9681
Epoch 4/10
28/28 [==============================] - 4s 154ms/step - loss: 0.0598
- accuracy: 0.9841 - val_loss: 0.0738 - val_accuracy: 0.9784
Epoch 5/10
28/28 [==============================] - 4s 156ms/step - loss: 0.0449
- accuracy: 0.9863 - val_loss: 0.0745 - val_accuracy: 0.9784
Epoch 6/10
28/28 [==============================] - 4s 156ms/step - loss: 0.0389
- accuracy: 0.9892 - val_loss: 0.0739 - val_accuracy: 0.9772
```

```
<keras.callbacks.History at 0x7f610ed77510>
```

```
#Testing The Model
print("Accuracy of the model on Testing Data is - " ,
model.evaluate(X_test,y_test)[1]*100 , "%")
```

```
25/25 [==============================] - 0s 15ms/step - loss: 0.0700 -
accuracy: 0.9768
Accuracy of the model on Testing Data is -  97.68041372299194 %
```