In [ ]:

```
# Baseline Code

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import seaborn as sns
from sklearn.svm import SVC



df = pd.read_csv('spam.tsv', sep='\t')

df.head()

df.count()

df.isna().sum()

df.describe()

df['label'].value_counts()

ham = df[df['label'] == 'ham']
spam = df[df['label'] == 'spam']

ham.shape, spam.shape

ham = ham.sample(spam.shape[0])

data = ham.append(spam, ignore_index=True)

X_train, X_test, y_train, y_test =  train_test_split(data['message'], data['label'], test_size = 0.3, random_st

X_train.shape

X_test.shape

classifier = Pipeline([("tfidf", TfidfVectorizer()) , ("classifier", RandomForestClassifier(n_estimators=100))]

classifier.fit(X_train, y_train)

y_pred1 = classifier.predict(X_test)

print(classification_report(y_test, y_pred1))

conf_mat1 = confusion_matrix(y_test, y_pred1)

print("Confusion Matrix:")
print(conf_mat1)


svm = Pipeline([("tfidf", TfidfVectorizer()) , ("classifier", SVC(C = 100, gamma='auto'))])

svm.fit(X_train, y_train)

y_pred2 = svm.predict(X_test)

print(classification_report(y_test, y_pred2))

conf_mat2 = confusion_matrix(y_test, y_pred2)

print("Confusion Matrix:")
print(conf_mat2)
```

In [ ]:

```
# Advance Code

import numpy as np
import pandas as pd
import matplotlib.pyplot as pit

from sklearn.model_selection import train_test_split

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Input, GlobalMaxPooling1D
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Embedding
from tensorflow.keras.layers import LSTM, Embedding
from tensorflow.keras.models import Model


df = pd.read_csv('spam.csv', encoding= 'ISO-8859-1')

df.head()

df = df.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)

df.columns = ['labels', 'data']

df['b_labels'] = df['labels'].map({'ham': 0, 'spam': 1})
y = df['b_labels'].values

x_train, x_test, y_train, y_test = train_test_split(df['data'], y, test_size = 0.33)

max_vocab_size = 20000
tokenizer = Tokenizer (num_words=max_vocab_size)
tokenizer.fit_on_texts(x_train)
sequences_train = tokenizer.texts_to_sequences (x_train)
sequences_test = tokenizer.texts_to_sequences (x_test)

word2idx = tokenizer.word_index
V = len(word2idx)
print ('Total number of unique tokens are: %s' % V)

data_train = pad_sequences(sequences_train)
print('shape of data train tensor:', data_train.shape)

T = data_train.shape[1]

print(T)

data_test = pad_sequences(sequences_test, maxlen=T)

print('Shape of data test tensor:', data_test.shape)

D = 20    # This is a hyperparameter, any vector size can be choosen.

# Input layer
i = Input(shape=(T,))    # Input layer takes in sequences of integers and returns word vectors

# Embedding layer
x = Embedding(V + 1, D)(i)

# CNN Layer
x = Conv1D(32, 3, activation='relu')(x)
x = MaxPooling1D(3)(x)

# Second CNN Layer
x = Conv1D(64, 3, activation='relu')(x)
x = MaxPooling1D(3)(x)

# Third CNN Layer
x = Conv1D(128, 3, activation='relu')(x)
x = GlobalMaxPooling1D()(x)

# Dense layer
x = Dense (1, activation='sigmoid')(x)

model = Model(i, x)


r = model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

r = model.fit (x = data_train, y = y_train, epochs = 5, validation_data=(data_test, y_test))


D1 = 20    # This is a hyperparameter, any vector size can be choosen.

# hidden state vectorize size
M = 15

# Input layer
i1 = Input(shape=(T,))    # Input layer takes in sequences of integers and returns word vectors

# Embedding layer
x1 = Embedding(V + 1, D1)(i1)

x1 = LSTM(M, return_sequences=True) (x1)
x1 = GlobalMaxPooling1D()(x1)

# Dense layer
x1 = Dense (1, activation='sigmoid')(x1)  # using activation 'sigmoid' because of it binary classification prob

model = Model(i1, x1)

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

r1 = model.fit (x = data_train, y = y_train, epochs = 10, validation_data=(data_test, y_test))
```