```
# import libraries
import tensorflow as tf
import pandas as pd
from tensorflow import keras

import tensorflow_datasets as tfds
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)


# get data files
!wget https://cdn.freecodecamp.org/project-data/sms/train-data.tsv
!wget https://cdn.freecodecamp.org/project-data/sms/valid-data.tsv

train_file_path = "train-data.tsv"
test_file_path = "valid-data.tsv"


# Reading Data & Setting Column Labels
train_data = pd.read_table(train_file_path, header=None, names=["spam", "message"])
test_data = pd.read_table(test_file_path, header=None, names=["spam", "message"])

train_data.head()


# Converting Categorical Data to Integers
train_data["spam"] = train_data["spam"].replace({"ham": 0, "spam": 1})
test_data["spam"] = test_data["spam"].replace({"ham": 0, "spam": 1})
train_data["spam"].value_counts()


# Modeling HamOrSpam Data
import seaborn as sns

sns.countplot(x='spam', data=train_data)
plt.show()


# Calculating HamOrSpam & Creating Data Frame
ham_msg = train_data[train_data["spam"] == 0]
neg = ham_msg.shape[0]
print(f"negative: {neg}")

spam_msg = train_data[train_data["spam"] == 1]
pos = spam_msg.shape[0]
print(f"positive: {pos}")

total = neg + pos
print(f"total: {total}")

balanced_data = train_data


# Cleaning Data: Removing Punctuation

import string
punctuations_list = string.punctuation
def remove_punctuations(text):
    temp = str.maketrans('', '', punctuations_list)
    return text.translate(temp)

balanced_data['message'] = balanced_data['message'].apply(lambda x: remove_punctuations(x))

# Perform the same pre-processing on test data as on training data.
test_data['message'] = test_data['message'].apply(lambda x: remove_punctuations(x))

balanced_data.head()
```

```
# Cleaning Data: Removing Stop Words

import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')

stop_words = stopwords.words('english')
new_stopwords = ["u", "ur"]
stop_words.extend(new_stopwords)

def remove_stopwords(text):
    imp_words = []

    # Storing the important words
    for word in str(text).split():
        word = word.lower()

        if word not in stop_words:
            imp_words.append(word)

    output = " ".join(imp_words)

    return output

balanced_data['message'] = balanced_data['message'].apply(lambda text: remove_stopwords(text))

# Perform the same pre-processing on test data as on training data.
test_data['message'] = test_data['message'].apply(lambda text: remove_stopwords(text))

balanced_data.head()


# Displaying Data as WordCloud

from wordcloud import WordCloud
def plot_word_cloud(data, typ):
    email_corpus = " ".join(data['message'])

    plt.figure(figsize=(7, 7))

    wc = WordCloud(
        background_color='black',
        max_words=100,
        width=800,
        height=400,
        collocations=False
    ).generate(email_corpus)

    plt.imshow(wc, interpolation='bilinear')
    plt.title(f'WordCloud for {typ} emails', fontsize=15)
    plt.axis('off')
    plt.show()

plot_word_cloud(balanced_data[balanced_data['spam'] == 0], typ='Non-Spam')
plot_word_cloud(balanced_data[balanced_data['spam'] == 1], typ='Spam')


# Removing Spam Classification Column

train_labels = balanced_data.pop("spam")
test_labels = test_data.pop("spam")

balanced_data


# Converting Message Column into Numpy Array

train_X = balanced_data["message"].to_numpy()
train_Y = train_labels.to_numpy()
test_X = test_data["message"].to_numpy()
test_Y = test_labels.to_numpy()
train_X
```

```
# Tokenizing & Padding Text

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

# Tokenize the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_X)

# Convert text to sequences
train_sequences = tokenizer.texts_to_sequences(train_X)
test_sequences = tokenizer.texts_to_sequences(test_X)

# Pad sequences to have the same length
max_len = 100

train_sequences = pad_sequences(
    train_sequences,
    maxlen=max_len,
    padding='post',
    truncating='post'
)
test_sequences = pad_sequences(
    test_sequences,
    maxlen=max_len,
    padding='post',
    truncating='post'
)
print(train_sequences[0])


# Build the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(
        input_dim=len(tokenizer.word_index) + 1,
        output_dim=32,
        mask_zero=True,
    ),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.LSTM(16),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Print the model summary
model.summary()


# Compiling Model

model.compile(
    loss = tf.keras.losses.BinaryCrossentropy(),
    metrics=['accuracy'],
    optimizer= tf.keras.optimizers.Adam(learning_rate=0.01, beta_1=0.9, beta_2=0.999)
)
```

```
# Setting up Early Stopping & Reducing Learning Rate to reduce Overfitting

from keras.callbacks import EarlyStopping, ReduceLROnPlateau
# callback
es = EarlyStopping(
    patience=4,
    monitor = 'val_accuracy',
    restore_best_weights = True
)

lr = ReduceLROnPlateau(
    patience = 2,
    monitor = 'val_loss',
    factor = 0.2,
    verbose = 1
)

# Weights
weight_for_0 = (1 / neg) * (total / 2.0)
weight_for_1 = (1 / pos) * (total / 2.0)

class_weight = {0: weight_for_0, 1: weight_for_1}

# Training
history = model.fit(
    x=train_sequences,
    y=train_Y,
    validation_split=0.1,
    epochs=20,
    batch_size=32,
    callbacks = [lr, es],
    class_weight=class_weight
)


# Displaying Model Accuracy over Epochs

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()


# Evaluating Model

test_loss, test_accuracy = model.evaluate(test_sequences, test_Y)
print('Test Loss :',test_loss)
print('Test Accuracy :',test_accuracy)


# function to predict messages based on model
# (should return list containing prediction and label, ex. [0.008318834938108921, 'ham'])

intTolabel = {0: "ham", 1: "spam"}

def predict_message(pred_text):
  pred_text = remove_punctuations(pred_text)
  pred_text = remove_stopwords(pred_text)

  sequence = tokenizer.texts_to_sequences([pred_text])
  sequence = pad_sequences(sequence, maxlen=max_len)

  prediction = model.predict(sequence)[0]

  label = "ham"
  if prediction >= 0.5:
      label = "spam"

  print((prediction[0], label))
  return (prediction[0], label)

pred_text = "how are you doing today?"

prediction = predict_message(pred_text)
```

```
# Run this cell to test your function and model. Do not modify contents.

def test_predictions():
  test_messages = ["how are you doing today",
                   "sale today! to stop texts call 98912460324",
                   "i dont want to go. can we try it a different day? available sat",
                   "our new mobile video service is live. just install on your phone to start watching.",
                   "you have won £1000 cash! call to claim your prize.",
                   "i'll bring it tomorrow. don't forget the milk.",
                   "wow, is your arm alright. that happened to me one time too"
                  ]

  test_answers = ["ham", "spam", "ham", "spam", "spam", "ham", "ham"]
  passed = True
```