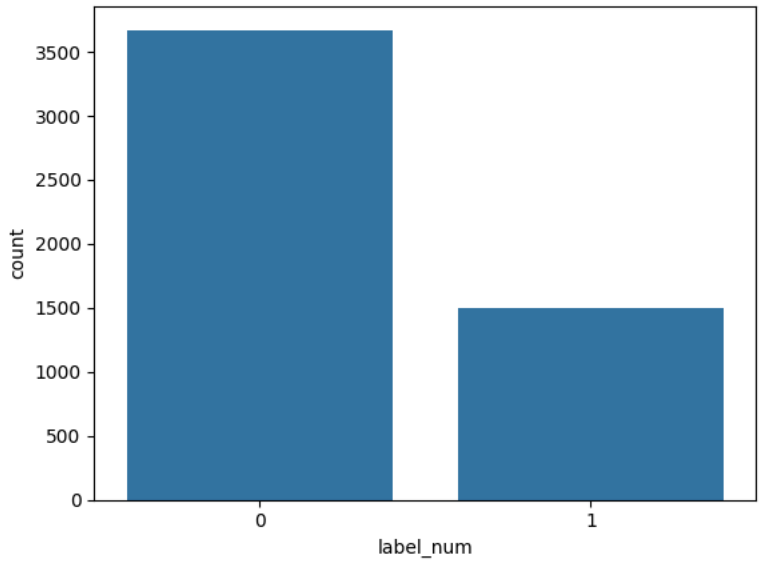
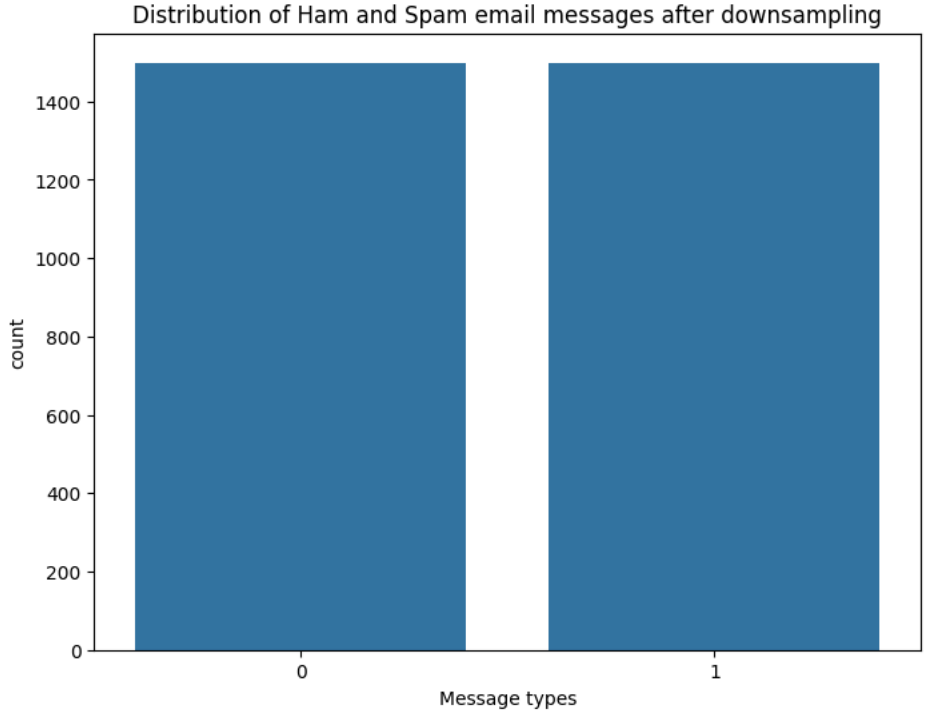
Start coding or generate with AI.

```
# Importing necessary libraries for EDA
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import string
import nltk
from nltk.corpus import stopwords
from wordcloud import WordCloud
nltk.download('stopwords')

# Importing libraries necessary for Model Building and Training
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping, ReduceLROnPlateau

import warnings
warnings.filterwarnings('ignore')

data = pd.read_csv('/content/spam_ham_dataset (1).csv')

data.head(10)

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

ham_msg = data[data.label_num == 0]
spam_msg = data[data.label_num == 1]
ham_msg = ham_msg.sample(n=len(spam_msg),
            random_state=42)

# Plotting the counts of down sampled dataset
# Use pd.concat to combine the DataFrames
balanced_data = pd.concat([ham_msg, spam_msg]).reset_index(drop=True)

plt.figure(figsize=(8, 6))
sns.countplot(data = balanced_data, x='label_num')
plt.title('Distribution of Ham and Spam email messages after downsampling')
plt.xlabel('Message types')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```




```
Text(0.5, 0, 'Message types')
```




```
balanced_data['text'] = balanced_data['text'].str.replace('Subject','')
balanced_data.head()
```

	Unnamed: 0	label	text	label_num
0	3444	ham	: conoco - big cowboy\r\ndarren :\r\ni ' m not...	0
1	2982	ham	: feb 01 prod : sale to teco gas processing\r\...	0
2	2711	ham	: california energy crisis\r\ncalifornia ⁇ , s...	0
3	3116	ham	: re : nom / actual volume for april 23 rd\r\n...	0
4	1314	ham	: eastrans nomination changes effective 8 / 2 ...	0

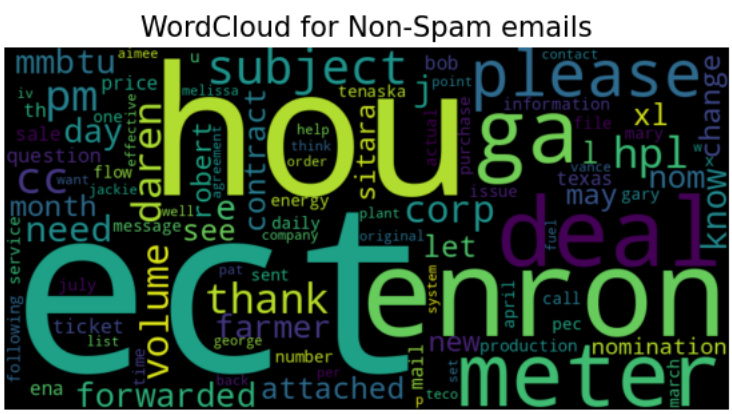
```
punctuations_list = string.punctuation
def remove_punctuations(text):
    temp = str.maketrans('', '', punctuations_list)
    return text.translate(temp)

balanced_data['text']= balanced_data['text'].apply(lambda x: remove_punctuations(x))
balanced_data.head()
```

	Unnamed: 0	label	text	label_num
0	3444	ham	conoco big cowboy\r\ndarren \r\ni m not sur...	0
1	2982	ham	feb 01 prod sale to teco gas processing\r\ns...	0
2	2711	ham	california energy crisis\r\ncalifornia ï s p...	0
3	3116	ham	re nom actual volume for april 23 rd\r\nwe ...	0
4	1314	ham	eastrans nomination changes effective 8 2 0...	0

```
def remove_stopwords(text):
    stop_words = stopwords.words('english')

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

	Unnamed: 0	label	text	label_num
0	3444	ham	conoco big cowboy darren sure help know else a...	0
1	2982	ham	feb 01 prod sale teco gas processing sale deal...	0
2	2711	ham	california energy crisis california ï power cr...	0
3	3116	ham	nom actual volume april 23 rd agree eileen pon...	0
4	1314	ham	eastrans nomination changes effective 8 2 00 p...	0

```
def plot_word_cloud(data, typ):
    email_corpus = " ".join(data['text'])

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




```
#train test split
train_X, test_X, train_Y, test_Y = train_test_split(balanced_data['text'],
                                                    balanced_data['label_num'],
                                                    test_size = 0.2,
                                                    random_state = 42)



# 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 # maximum sequence length
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')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-b532902eb349> in <cell line: 2>()
      1 # Tokenize the text data
----> 2 tokenizer = Tokenizer()
      3 tokenizer.fit_on_texts(train_X)
      4
      5 # Convert text to sequences

NameError: name 'Tokenizer' is not defined
```

```
# Build the model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Embedding(input_dim=len(tokenizer.word_index) + 1,
                                    output_dim=32,
                                    input_length=max_len))
model.add(tf.keras.layers.LSTM(16))
model.add(tf.keras.layers.Dense(32, activation='relu'))
```

```
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

# Print the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 32)           1274912

 lstm (LSTM)                 (None, 16)                3136

 dense (Dense)               (None, 32)                544

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 1278625 (4.88 MB)
Trainable params: 1278625 (4.88 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(loss = tf.keras.losses.BinaryCrossentropy(from_logits = True),
            metrics = ['accuracy'],
            optimizer = 'adam')



es = EarlyStopping(patience=3,
                monitor = 'val_accuracy',
                restore_best_weights = True)

lr = ReduceLROnPlateau(patience = 2,
                    monitor = 'val_loss',
                    factor = 0.5,
                    verbose = 0)


# Train the model
history = model.fit(train_sequences, train_Y,
                    validation_data=(test_sequences, test_Y),
                    epochs=20,
                    batch_size=32,
                    callbacks = [lr, es]
                )
```

```
Epoch 1/20
75/75 [==============================] - 10s 87ms/step - loss: 0.5971 - accuracy: 0.6714 - val_loss: 0.2215 - val_accuracy: 0.9483
Epoch 2/20
75/75 [==============================] - 4s 55ms/step - loss: 0.1595 - accuracy: 0.9637 - val_loss: 0.1675 - val_accuracy: 0.9600 -
Epoch 3/20
75/75 [==============================] - 4s 54ms/step - loss: 0.1169 - accuracy: 0.9746 - val_loss: 0.1732 - val_accuracy: 0.9600 -
Epoch 4/20
75/75 [==============================] - 5s 72ms/step - loss: 0.1061 - accuracy: 0.9775 - val_loss: 0.1871 - val_accuracy: 0.9567 -
Epoch 5/20
75/75 [==============================] - 4s 54ms/step - loss: 0.0922 - accuracy: 0.9812 - val_loss: 0.1809 - val_accuracy: 0.9583 -
```

```
# Train the model
history = model.fit(train_sequences, train_Y,
                    validation_data=(test_sequences, test_Y),
                    epochs=20,
                    batch_size=32,
                    callbacks = [lr, es]
                )
```

```
Epoch 1/20
75/75 [==============================] - 4s 56ms/step - loss: 0.1177 - accuracy: 0.9746 - val_loss: 0.1696 - val_accuracy: 0.9600 -
Epoch 2/20
75/75 [==============================] - 5s 66ms/step - loss: 0.1170 - accuracy: 0.9746 - val_loss: 0.1706 - val_accuracy: 0.9600 -
Epoch 3/20
75/75 [==============================] - 6s 75ms/step - loss: 0.1068 - accuracy: 0.9771 - val_loss: 0.1794 - val_accuracy: 0.9583 -
Epoch 4/20
75/75 [==============================] - 5s 63ms/step - loss: 0.1023 - accuracy: 0.9783 - val_loss: 0.1924 - val_accuracy: 0.9550 -
```

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