In [1]:
```
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 nltk import word_tokenize
nltk.download('stopwords')
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')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Acer\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

In [6]:
```
data = pd.read_csv('spam.csv')
data.head(10)
```

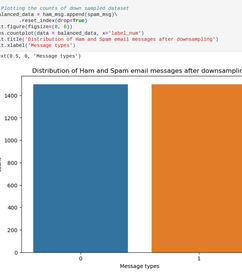
Out[6]:

	Unnamed: 0	label	text	label_num
0	605	ham	Subject: enron methanol ; meter # : 988291\r\n...	0
1	2349	ham	Subject: hpl nom for january 9 , 2001\r\n(see...	0
2	3624	ham	Subject: neon retreat\r\nho ho ho , we ' re ar...	0
3	4685	spam	Subject: photoshop , windows , office . cheap ...	1
4	2030	ham	Subject: re : indian springs\r\nthis deal is t...	0
5	2949	ham	Subject: vhronline web address change\r\nthis ...	0
6	2793	ham	Subject: spring savings certificate - take 30 ...	0
7	4185	spam	Subject: looking for medication ? we ' re the ...	1
8	2641	ham	Subject: noms / actual flow for 2 / 20\r\nwe a...	0
9	1870	ham	Subject: nominations for oct . 21 - 23 , 2000\...	0

In [7]:
```
data.shape
```

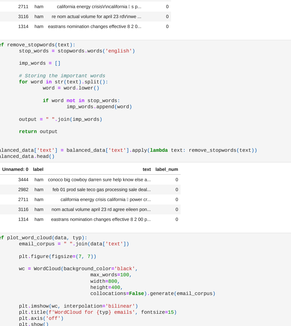
Out[7]:
```
(5171, 4)
```

In [8]:
```
sns.countplot(x='label_num', data=data)
plt.show()
```



In [9]:
```
# Downsampling to balance the dataset
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
balanced_data = ham_msg.append(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')
```

Out[9]:
```
Text(0.5, 0, 'Message types')
```



In [10]:
```
balanced_data['text'] = balanced_data['text'].str.replace('Subject', '')
balanced_data.head()
```

Out[10]:

	Unnamed: 0	label	text	label_num
0	3444	ham	: conoco - big cowboy\r\ndarren: \r\n Yes , I'm not...	0
1	2982	ham	: feb 01 prod : sale to tesco gas processing\r\n...	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

In [11]:
```
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()
```

Out[11]:

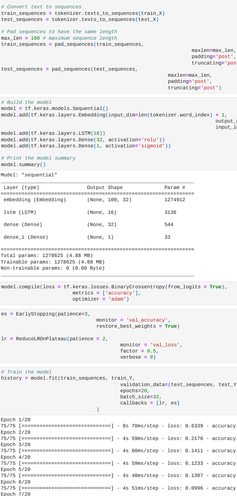
	Unnamed: 0	label	text	label_num
0	3444	ham	conoco big cowboy\r\ndarren\r\n Yes I'm not su...	0
1	2982	ham	feb 01 prod sale to teco gas processing\r\nIs...	0
2	2711	ham	california energy crisis\r\ncalifornia s p...	0
3	3116	ham	re nom actual volume for april 23 rd\r\nIner ...	0
4	1314	ham	eastrans nomination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

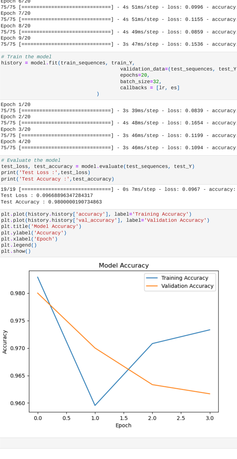
In [12]:
```
def remove_stopwords(text):
    stop_words = stopwords.words('english')

    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()
```

Out[12]:

	Unnamed: 0	label	text	label_num
0	3444	ham	conoco big cowboy\r\ndarren\r\n Yes I'm not su...	0
1	2982	ham	feb 01 prod sale teco gas processing\r\nIs deal...	0
2	2711	ham	california energy crisis\r\ncalifornia (power cr...	0
3	3116	ham	nom actual volume april 23 rd\r\nIner enron pon...	0
4	1314	ham	eastrans nomination changes effective 8 2 00 p...	0

In [13]:
```
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('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')
```





In [14]:
```
#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)
```

In [15]:
```
# 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')
```

In [16]:
```
# 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)
_________________________________________________________________
```

In [17]:
```
model.compile(loss = tf.keras.losses.BinaryCrossentropy(from_logits = True),
              metrics = ['accuracy'],
              optimizer = 'adam')
```

In [18]:
```
es = EarlyStopping(patience=3,
                   monitor = 'val_accuracy',
                   restore_best_weights = True)

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

In [19]:
```
# 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 [==============================] - 8s 79ms/step - loss: 0.6339 - accuracy: 0.6493 - val_loss: 0.3189 - val_accuracy: 0.9600 - lr: 5.0000e-04
Epoch 2/20
75/75 [==============================] - 4s 59ms/step - loss: 0.2170 - accuracy: 0.9462 - val_loss: 0.1568 - val_accuracy: 0.9667 - lr: 5.0000e-04
Epoch 3/20
75/75 [==============================] - 4s 60ms/step - loss: 0.1411 - accuracy: 0.9683 - val_loss: 0.1547 - val_accuracy: 0.9633 - lr: 5.0000e-04
Epoch 4/20
75/75 [==============================] - 4s 59ms/step - loss: 0.1233 - accuracy: 0.9725 - val_loss: 0.1622 - val_accuracy: 0.9617 - lr: 5.0000e-04
Epoch 5/20
75/75 [==============================] - 4s 48ms/step - loss: 0.1307 - accuracy: 0.9691 - val_loss: 0.1402 - val_accuracy: 0.9683 - lr: 5.0000e-04
Epoch 6/20
75/75 [==============================] - 4s 51ms/step - loss: 0.0996 - accuracy: 0.9779 - val_loss: 0.1146 - val_accuracy: 0.9750 - lr: 5.0000e-04
Epoch 7/20
75/75 [==============================] - 4s 51ms/step - loss: 0.1155 - accuracy: 0.9729 - val_loss: 0.1412 - val_accuracy: 0.9617 - lr: 5.0000e-04
Epoch 8/20
75/75 [==============================] - 4s 49ms/step - loss: 0.0859 - accuracy: 0.9821 - val_loss: 0.4893 - val_accuracy: 0.8850 - lr: 5.0000e-04
Epoch 9/20
75/75 [==============================] - 3s 47ms/step - loss: 0.1536 - accuracy: 0.9629 - val_loss: 0.1350 - val_accuracy: 0.9683 - lr: 5.0000e-04
```

In [20]:
```
# 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 [==============================] - 3s 39ms/step - loss: 0.0839 - accuracy: 0.9829 - val_loss: 0.0967 - val_accuracy: 0.9800 - lr: 5.0000e-04
Epoch 2/20
75/75 [==============================] - 4s 48ms/step - loss: 0.1054 - accuracy: 0.9595 - val_loss: 0.1165 - val_accuracy: 0.9700 - lr: 5.0000e-04
Epoch 3/20
75/75 [==============================] - 3s 46ms/step - loss: 0.1199 - accuracy: 0.9708 - val_loss: 0.1279 - val_accuracy: 0.9633 - lr: 5.0000e-04
Epoch 4/20
75/75 [==============================] - 3s 46ms/step - loss: 0.1094 - accuracy: 0.9733 - val_loss: 0.1347 - val_accuracy: 0.9617 - lr: 2.5000e-04
```

In [21]:
```
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_sequences, test_Y)
print('Test Loss :',test_loss)
print('Test Accuracy :',test_accuracy)
```

```
30/30 [==============================] - 0s 7ms/step - loss: 0.0967 - accuracy: 0.9800
Test Loss : 0.09668898582458496
Test Accuracy : 0.9800000190734863
```

In [22]:
```
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()
```


In [ ]: