Detecting Spam Emails

Spam emails are unwanted emails sent in bulk. Detecting spam mails helps the users to avoid clutters in their inbox.

In this project, we are going to built a spam detection model to identify whether an email is spam or not using a popular deeplearning library called, Tensorflow.

Import Libraries

In [81]:
```
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')
nltk.download('wordnet') # Download wordnet for lemmatization

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

Load the dataset

In [82]:
```
data = pd.read_csv('/content/spam_ham_dataset.csv')
data.head()
```

Out[82]:

	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

In [83]:
```
display(data.shape)
display(data.describe())
display(data.info())
display(data.isna().sum())
```

```
(5171, 4)
```

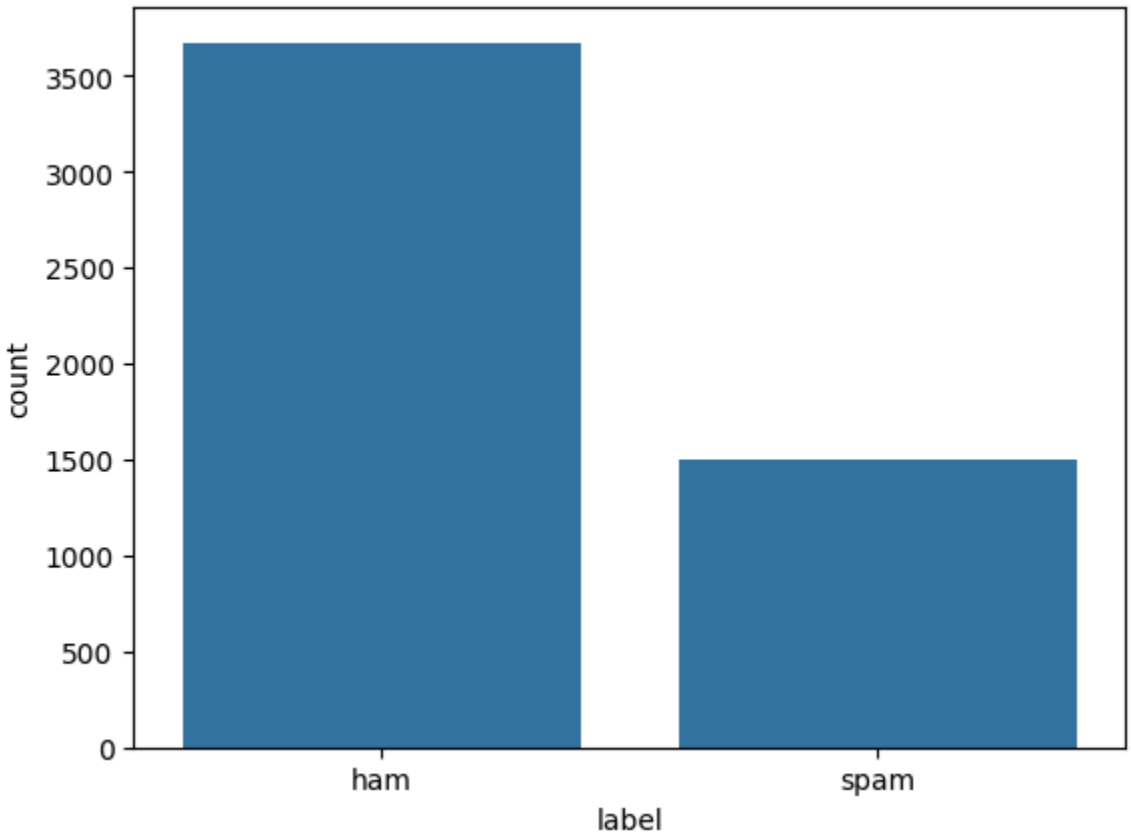
	Unnamed: 0	label_num
count	5171.000000	5171.000000
mean	2585.000000	0.289886
std	1492.883452	0.453753
min	0.000000	0.000000
25%	1292.500000	0.000000
50%	2585.000000	0.000000
75%	3877.500000	1.000000
max	5170.000000	1.000000

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5171 entries, 0 to 5170
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Unnamed: 0  5171 non-null   int64 
 1   label       5171 non-null   object
 2   text        5171 non-null   object
 3   label_num   5171 non-null   int64 
dtypes: int64(2), object(2)
memory usage: 161.7+ KB
None
```

	0
Unnamed: 0	0
label	0
text	0
label_num	0

dtype: int64

In [84]:
```
sns.countplot(x = 'label', data=data)
plt.show()
```



Balance the Dataset

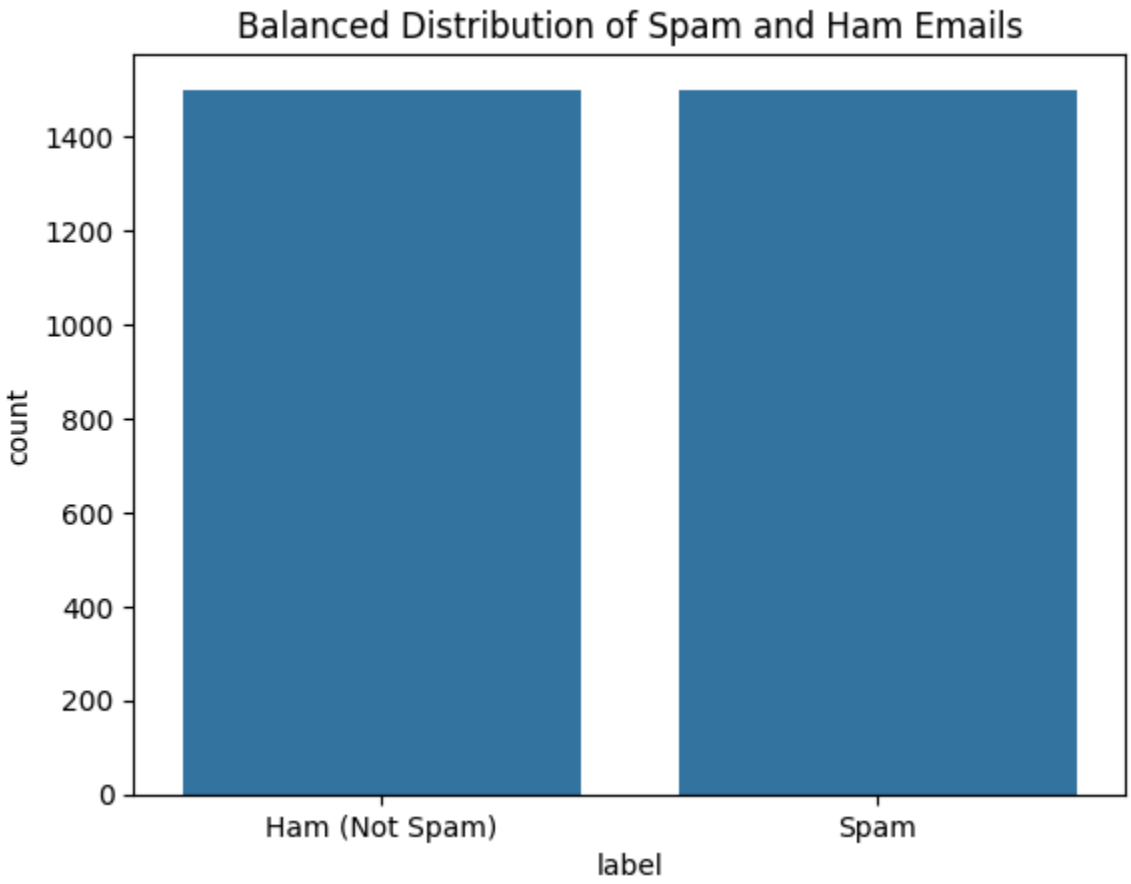
We can see there is an imbalance in the dataset, where the number of ham emails are more than spam emails. To address the imbalance we will downsample the majority class, ham to match the spam.

In [85]:
```
ham_msg = data[data['label'] == 'ham']
spam_msg = data[data['label'] == 'spam']
```

```
# Downsample the ham emails
ham_msg_balanced = ham_msg.sample(n= len(spam_msg), random_state=42)

# combine balanced data
balanced_data = pd.concat([ham_msg_balanced, spam_msg]).reset_index(drop=True)

# Visualize the balanced dataset
sns.countplot(x='label', data=balanced_data)
plt.title("Balanced Distribution of Spam and Ham Emails")
plt.xticks(ticks=[0, 1], labels=['Ham (Not Spam)', 'Spam'])
plt.show()
```



Clean the Text

Textual data requires preprocessing before feeding into ML models. Common steps include, removing stop words, removing punctuations, and performing stemming/ lemmatization. We will perform

- Removal of stop words
- Removal of punctuation
- Stemming/Lemmatization

In [86]:
```
balanced_data['text'] = balanced_data['text'].str.replace('Subject', '')
```

```
balanced_data.head()
```

Out[86]:

	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

In [87]:

```
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_punctuatic
balanced_data.head()
```

Out[87]:

	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 ⁇ 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

In [88]:

```
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_stopwc
balanced_data.head()
```

Out[88]:

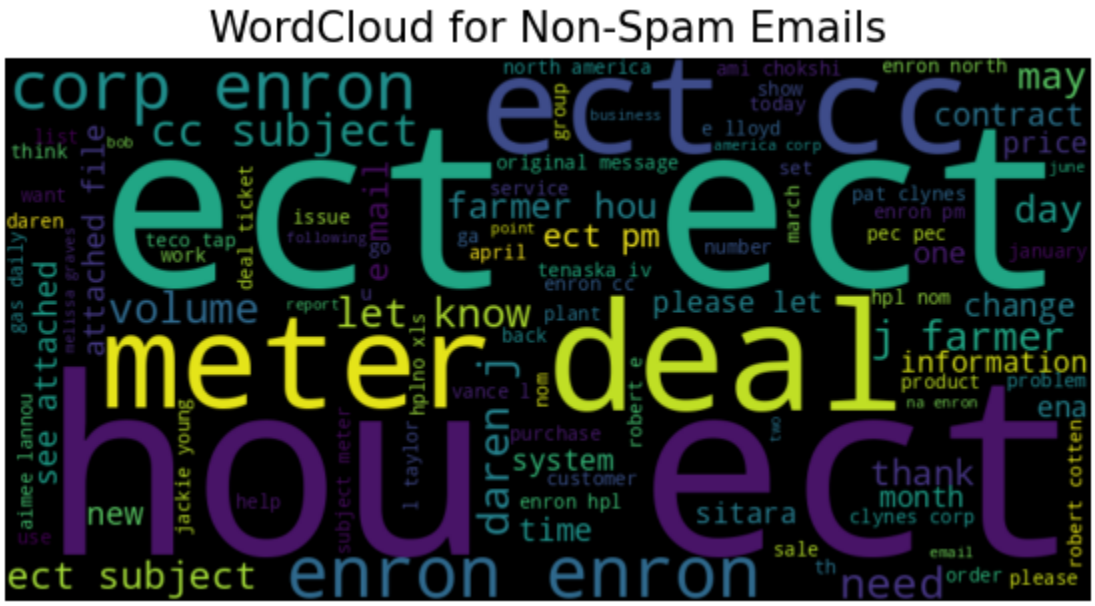
	Unnamed: 0	label	text	label_num
0	3444	ham	conoco big cowboy darren sure help know else a...	0
1	2982	ham	feb 01 prod sale teco gas processing sale deal...	0
2	2711	ham	california energy crisis california ⚠ power cr...	0
3	3116	ham	nom actual volume april 23 rd agree eileen pon...	0
4	1314	ham	eastrans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

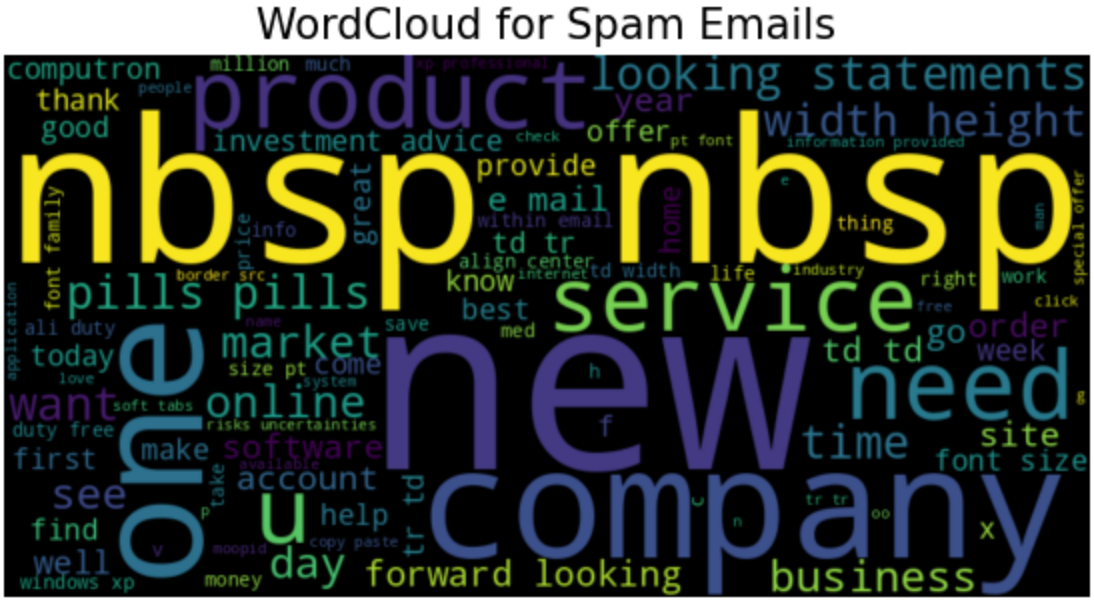
Visualization of Word Cloud

A word cloud is a text visualisation tool which helps us to give insights about the most frequent words present in the corpus of the data.

In [89]:
```
def plot_word_cloud(data, typ):
    email_corpus = " ".join(data['text'])
    wc = WordCloud(background_color='black', max_words=100, width=800, height=
    plt.figure(figsize=(7, 7))
    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'] == 'ham'], typ='Non-Spam'
plot_word_cloud(balanced_data[balanced_data['label'] == 'spam'], typ='Spam')
```





Tokenization and Padding

Machine learning model works with numbers. So we need to convert the text data into numerical vectors using Tokenization and Padding.

Tokenization : Converts each word into a unique integer.

Padding: Ensures that all text sequences have the same length, making them compatible with the model.

In [90]:
```
train_X, test_X, train_Y, test_Y = train_test_split(
    balanced_data['text'], balanced_data['label'], test_size=0.2, random_state
)

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_X)

train_sequences = tokenizer.texts_to_sequences(train_X)
test_sequences = tokenizer.texts_to_sequences(test_X)

max_len = 100  # Maximum sequence length
train_sequences = pad_sequences(train_sequences, maxlen=max_len, padding='post
test_sequences = pad_sequences(test_sequences, maxlen=max_len, padding='post',

train_Y = (train_Y == 'spam').astype(int)
test_Y = (test_Y == 'spam').astype(int)
```

Define the Model

We will build a deep learning model using a Sequential architecture. This model will include:

Embedding Layer: Learns vector representations of words.

LSTM Layer: Captures patterns in sequences.

Fully Connected Layer: Extracts relevant features.

Output Layer: Predicts whether an email is spam or not.

In [91]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(input_dim=len(tokenizer.word_index) + 1, output_
    tf.keras.layers.LSTM(16),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')  # Output layer
])

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

model.summary()
```

Model: "sequential_10"

Layer (type)	Output Shape	Param #
embedding_10 (Embedding)	?	0 (unbuilt)
lstm_11 (LSTM)	?	0 (unbuilt)
dense_21 (Dense)	?	0 (unbuilt)
dense_22 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

Train the Model

We train the model using EarlyStopping and ReduceLROnPlateau callbacks. These callbacks help stop the training early if the model's performance doesn't improve

and reduce the learning rate to fine-tune the model.

In [92]:
```
es = EarlyStopping(patience=3, monitor='val_accuracy', restore_best_weights=Tr
lr = ReduceLROnPlateau(patience=2, monitor='val_loss', factor=0.5, verbose=0)

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 ━━━━━━━━━━━━━━━━━━━━ 7s 55ms/step - accuracy: 0.5564 - loss: 0.6899 - va
l_accuracy: 0.9350 - val_loss: 0.4101 - learning_rate: 0.0010
Epoch 2/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 69ms/step - accuracy: 0.9419 - loss: 0.2848 - va
l_accuracy: 0.9467 - val_loss: 0.2010 - learning_rate: 0.0010
Epoch 3/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 52ms/step - accuracy: 0.9470 - loss: 0.2043 - va
l_accuracy: 0.9567 - val_loss: 0.1753 - learning_rate: 0.0010
Epoch 4/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - accuracy: 0.9579 - loss: 0.1728 - va
l_accuracy: 0.9533 - val_loss: 0.1867 - learning_rate: 0.0010
Epoch 5/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 69ms/step - accuracy: 0.9579 - loss: 0.1690 - va
l_accuracy: 0.9650 - val_loss: 0.1530 - learning_rate: 0.0010
Epoch 6/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.9727 - loss: 0.1253 - va
l_accuracy: 0.9600 - val_loss: 0.1662 - learning_rate: 0.0010
Epoch 7/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.9787 - loss: 0.1000 - va
l_accuracy: 0.9600 - val_loss: 0.1710 - learning_rate: 0.0010
Epoch 8/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 65ms/step - accuracy: 0.9834 - loss: 0.0822 - va
l_accuracy: 0.9633 - val_loss: 0.1616 - learning_rate: 5.0000e-04
```

After training, we evaluate the model on the test data to measure its performance.

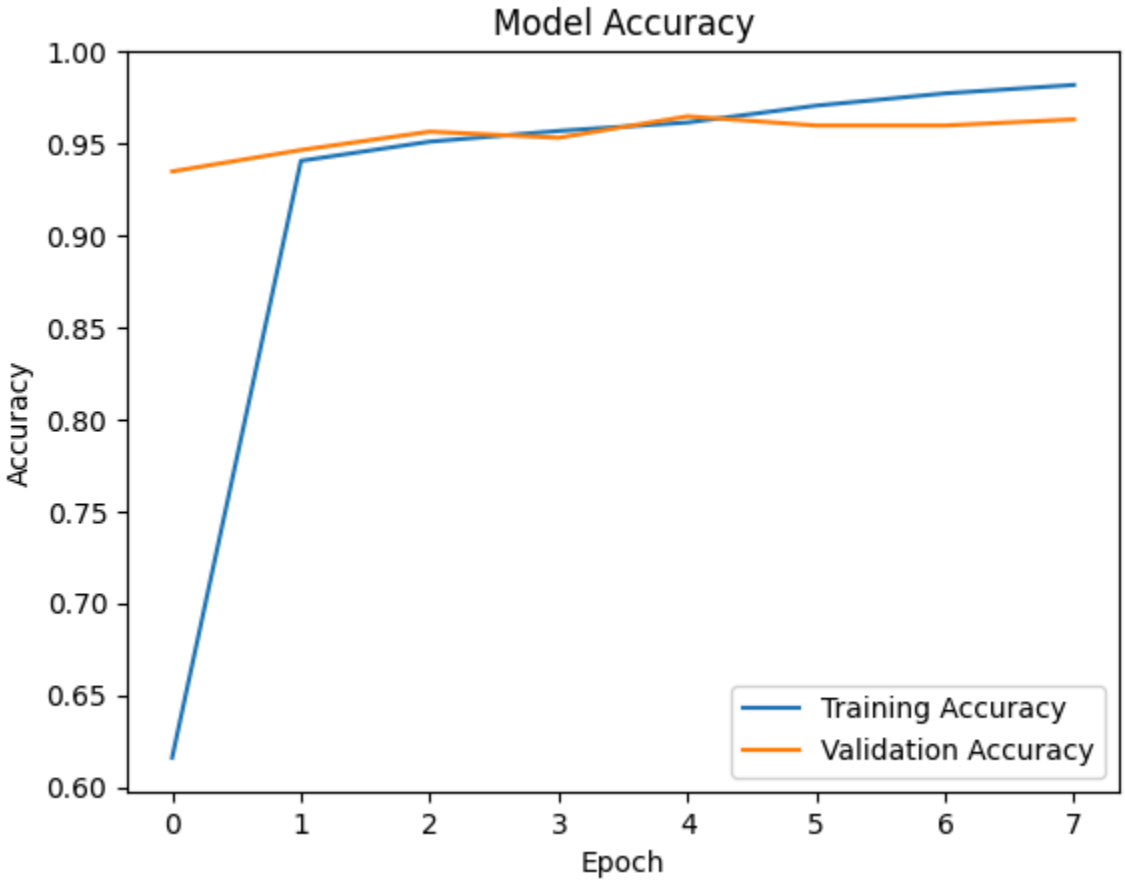
In [93]:
```
test_loss, test_accuracy = model.evaluate(test_sequences, test_Y)
print('Test Loss :',test_loss)
print('Test Accuracy :',test_accuracy)
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9703 - loss: 0.1370
Test Loss : 0.15300357341766357
Test Accuracy : 0.9649999737739563
```

Thus, the training accuracy turns out to be 97% which is quite satisfactory.

Having trained our model, we can plot a graph depicting the variance of training and validation accuracies with the no. of epochs.

In [94]:
```
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()
```



By following these steps, we have successfully built a machine learning model that can classify emails as spam or ham. With further optimization, this model can be fine-tuned to improve its performance even more.