```
import pandas as pd
import numpy as np
import sklearn
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split
```

```
df = pd.read_csv('/content/SPAM text message 20170820 - Data.csv')
df.head()
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
df.loc[df['Category'] == 'spam', 'Category'] = 1
df.loc[df['Category'] == 'ham', 'Category'] = 0
```

```
X = df.Message
y = df['Category'].astype('int')
```

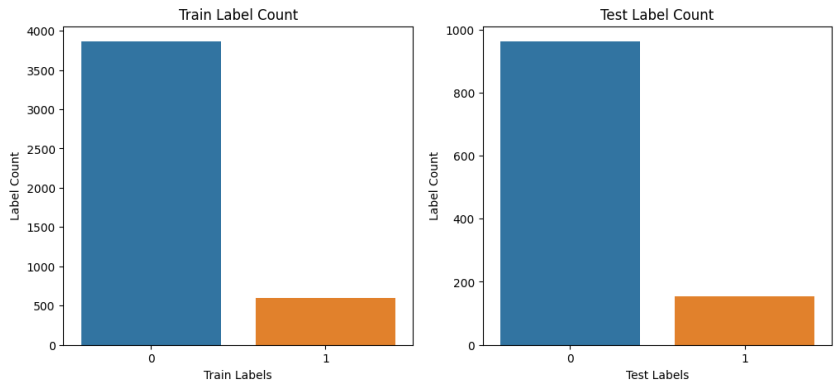
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

X_train.shape
```

```
(4457,)
```

```
fig, ax = plt.subplots(1,2, figsize=(12,5))
for idx, group in enumerate([('Train', y_train), ('Test', y_test)]):
    data = group[1].value_counts()
    sns.barplot(ax=ax[idx], x=data.index, y=data.values)
    ax[idx].set_title(f'{group[0]} Label Count')
    ax[idx].set_xlabel(f'{group[0]} Labels')
    ax[idx].set_ylabel('Label Count')

plt.show()
```



Description of the Dataset and what the Model should Predict

The dataset consists of two columns consisting of 'Category' and 'Message'. The message is going to contain text which has to be classified as whether it is spam or ham. This dataset is used to identify spam text in emails. The model should be able to predict whether the message is spam or not spam depending on the message in text.

```
import nltk
nltk.download('stopwords')

from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=list(stopwords))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

```
print('train size:', X_train.shape)
print(X_train.toarray()[:5])

print('\ntest size:', X_test.shape)
print(X_test.toarray()[:5])
```

```
train size: (4457, 7534)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

test size: (1115, 7534)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

```
from sklearn.naive_bayes import MultinomialNB

naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
import math
prior_p = sum(y_train == 1)/len(y_train)
print('prior spam:', prior_p, 'log of prior:', math.log(prior_p))

naive_bayes.class_log_prior_[1]
```

```
prior spam: 0.13327350235584473 log of prior: -2.015351853583911
-2.01535185358391
```

```
naive_bayes.feature_log_prob_
```

```
array([[-9.747493  , -9.747493  , -9.53591126, ..., -9.747493  ,
        -9.59780933, -9.57494494],
       [-8.14537582, -7.30051404, -9.19217512, ..., -8.94683138,
        -9.19217512, -9.19217512]])
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

pred = naive_bayes.predict(X_test)
```

```
print(confusion_matrix(y_test, pred))
```

```
[[962   0]
 [ 38 115]]
```

```
print('accuracy score: ', accuracy_score(y_test, pred))

print('\nprecision score (not spam): ', precision_score(y_test, pred, pos_label=0))
print('precision score (spam): ', precision_score(y_test, pred))

print('\nrecall score: (not spam)', recall_score(y_test, pred, pos_label=0))
print('recall score: (spam)', recall_score(y_test, pred))

print('\nf1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.9659192825112107

precision score (not spam):  0.962
precision score (spam):  1.0

recall score: (not spam) 1.0
recall score: (spam) 0.7516339869281046

f1 score:  0.8582089552238806
```

Naive Bayes Model Analysis

The Naive Bayes formula describes the probability of an event depending on the prior knowledge of the event given. The accuracy for this text classification on whether the message is spam or not spam. Naive Bayes does this by calculating the probability of whether a certain piece of text belongs to a certain category (spam/not spam) based on the frequency of terms and phrases in the text.

As you can see from the code above, the TF-IDF is used to calculate the term frequency which is used to categorize the text. Then a Naive Bayes model is run from sklearn on the training dataset in order to give the accuracy. The outputs given above on the accuracy score and precision score above is run on the test dataset.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, log_loss

classifier = LogisticRegression(solver='lbfgs', class_weight='balanced')
classifier.fit(X_train, y_train)

pred = classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
probs = classifier.predict_proba(X_test)
print('log loss: ', log_loss(y_test, probs))
```

```
accuracy score:  0.9757847533632287
precision score:  0.89375
recall score:  0.934640522875817
f1 score:  0.9137380191693291
log loss:  0.1627328555881471
```

Logistic Regression Model Analysis

Logistic Regression is widely used for blinary classification tasks, such as spam message classification. The goal of the model is to predict whether the message is spam or not. During the training process, the Logistic Regression model estimates the probability that a given text message is spam based on the features such as words/phrases that are in the message. The model is then used on the test dataset to see the performance. As you can see the Logistic Regression Model performs very well with an accurancy score of 97%.

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(solver='lbfgs', alpha=1e-5,
                   hidden_layer_sizes=(15, 2), random_state=1)
classifier.fit(X_train, y_train)
```

```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(15, 2), random_state=1,
              solver='lbfgs')
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score
pred = classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.9847533632286996
precision score:  0.9473684210526315
recall score:  0.9411764705882353
f1 score:  0.9442622950819671
```

Neural Network Model Analysis

The MLPClassifier (Multi-Layer Perceptron Classifier) is a machine learning algorithm that can be used for spam text classification. It is a type of neural network that consists of multiple layers of nodes, that are interconnected by weighted connections. To use this model, you would have to prepare the data similarly to the other two models.

Next a MLPClassifier model is trained using the train data and the optimization algorithm used in our model is Limited-Memory BFGS algorithm. Then the model is run on the testing data to get the prediction.

A benefit of the Neural Network approach compared to using a logistic regression model is that it can learn more complex nonlinear relationships between the features and class labels. However it also requires more computing resources and may be more prone to overfitting.

Based on our accuracies from the models run, the Neural Network approach is more accurate compared to the Logistic Regression and Naive Bayes approach.