In [1]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.metrics import confusion_matrix
```

In [2]:
```
#dataset is separated by tab, so we use seperator='\t'
data = pd.read_csv("./SMSSpamData", sep='\t', names=['label', 'message'])
```

## Label:

**Spam**: message is spam
**ham**: message is not spam

In [3]:
```
data.head()
```

Out[3]:

| | label | 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... |

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
label      5572 non-null object
message    5572 non-null object
dtypes: object(2)
memory usage: 87.2+ KB
```

**convert our labels to binary variables**, 0 to represent 'ham'(i.e. not spam) and 1 to represent 'spam'

In [5]:
```
#use '1' for spam and '0' for not spam
data['label'] = data.label.map({'ham':0, 'spam':1})
data.head()
```

Out[5]:

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

In [6]:
```
# split into training and testing sets
# USE from sklearn.model_selection import train_test_split to avoid seeing depre
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data['message'],
                                                    data['label'],
                                                    test_size =0.2,
                                                    random_state=1)


print('Number of rows in the total set: {}'.format(data.shape[0]))
print('Number of rows in the training set: {}'.format(X_train.shape[0]))
print('Number of rows in the test set: {}'.format(X_test.shape[0]))
```

```
Number of rows in the total set: 5572
Number of rows in the training set: 4457
Number of rows in the test set: 1115
```

## Frequency distribution

Our objective here is to convert this set of text to a frequency distribution matrix

**Note**

- The CountVectorizer method automatically converts all tokenized words to their lower case form so that it does not treat words like 'He' and 'he' differently. It does this using the lowercase parameter which is by default set to True.
- It also ignores all punctuation so that words followed by a punctuation mark (for example: 'hello!') are not treated differently than the same words not prefixed or suffixed by a punctuation mark (for example: 'hello').

In [7]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_vector = CountVectorizer()
```

**fit_transform( )**

Learn the vocabulary dictionary and return term-document matrix.

**transform( )**

Transform documents to document-term matrix.

In [8]:
```
# Fit the training data and then return the matrix
training_data = count_vector.fit_transform(X_train).toarray()

# Transform testing data and return the matrix. Note we are not fitting the test
testing_data = count_vector.transform(X_test).toarray()
```

In [9]:
```
frequency_matrix = pd.DataFrame(training_data,
                                columns = count_vector.get_feature_names())
frequency_matrix.head()
```

Out[9]:

| | 00 | 000 | 008704050406 | 0121 | 01223585236 | 01223585334 | 0125698789 | 02 | 0207 | 02072069400 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

5 rows × 7714 columns

In [10]:
```
testing_data
```

Out[10]:
```
array([[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],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [11]:
```
#Train the data
clf = LogisticRegression(random_state=0).fit(training_data, y_train)
```

In [12]:
```
#predict the value
predictions = clf.predict(testing_data)
```

In [13]:
```
predictions
```

Out[13]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

## Result

In [14]:
```
print('Accuracy score: ', format(accuracy_score(y_test, predictions)))
print('Precision score: ', format(precision_score(y_test, predictions)))
print('Recall score: ', format(recall_score(y_test, predictions)))
print('F1 score: ', format(f1_score(y_test, predictions)))
print('\nConfusion Matrix :\n', confusion_matrix(y_test, predictions))
```

```
Accuracy score:  0.989237668161435
Precision score:  0.9927007299270073
Recall score:  0.9251700680272109
F1 score:  0.9577464788732395

Confusion Matrix :
 [[967   1]
 [ 11 136]]
```