Dataset

https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

In [1]:
```
import numpy as np
import pandas as pd
import re
import nltk
import seaborn as sns
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,classification_report as cr
```

In [2]:
```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[2]:
```
True
```

In [3]:
```
# Reading the dataset
spam_text = pd.read_csv('SPAM text message Dataset.csv')
```

In [4]:
```
spam_text
```

Out[4]:

	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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

In [5]:
```
# Looking for one message example
spam_text['Message'].loc[4995]
```

Out[5]:
```
'My drive can only be read. I need to write'
```

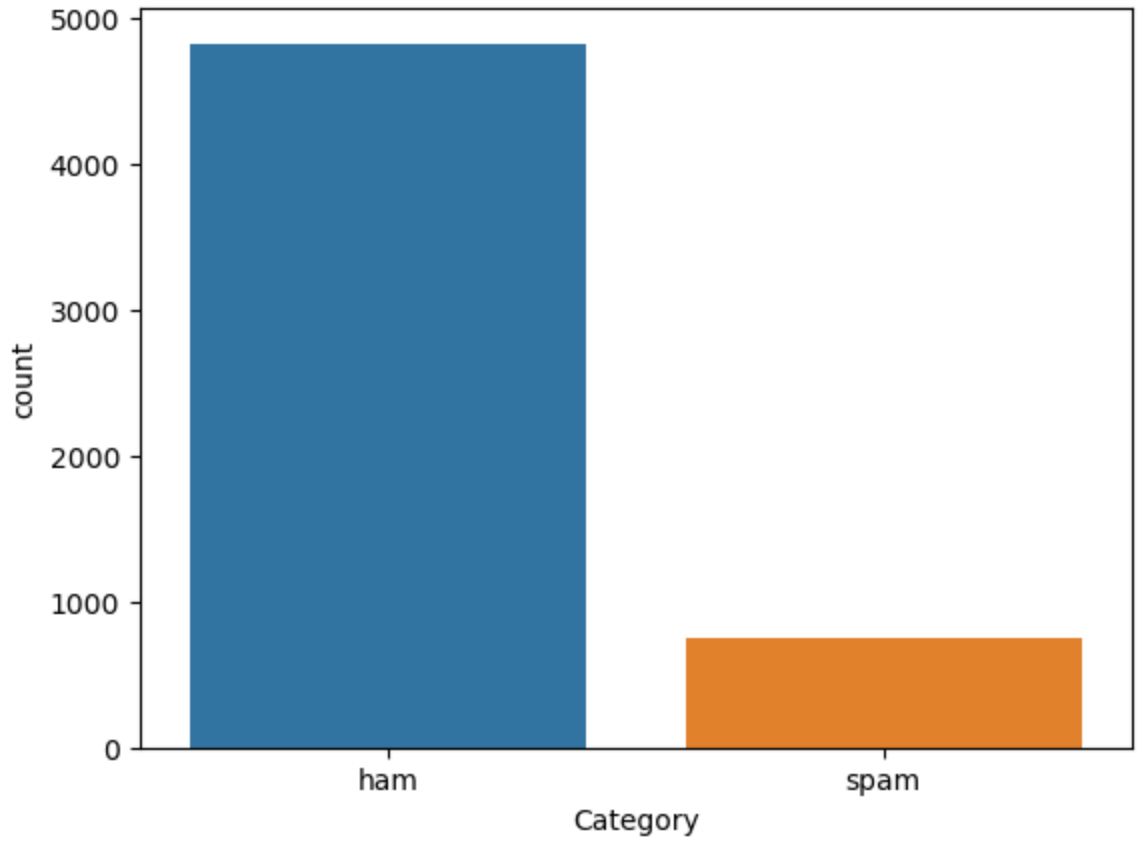
In [6]:
```
# Downloading Stop Words
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[6]:
```
True
```

In [7]:
```
sns.countplot(x='Category',data=spam_text)
```

Out[7]:
```
<Axes: xlabel='Category', ylabel='count'>
```



Implementing Stemming and removing stop words in the messages

In [8]:
```
# Calling portstemmer for the purpose of stemming the words in messages
ps = PorterStemmer()
```

In [9]:
```
corpus = []
for i in range(0,len(spam_text)):
    messages = re.sub('[^a-zA-Z]',' ',spam_text['Message'][i])
    messages = messages.lower()
    messages = messages.split()
    messages = [ps.stem(word) for word in messages if not word in stopwords.words('engli
    messages = ' '.join(messages)
    corpus.append(messages)
```

Applying BagOfWords (BOW) Method

In [10]:
```
# Creating a bag of words model
cv = CountVectorizer(max_features=2500)
x = cv.fit_transform(corpus).toarray()
```

In [11]:
```
# Converting the categorical values into dummy variables
y = pd.get_dummies(spam_text['Category'])
y = y.iloc[:,1].values
```

In [12]:
```
# spliting the data for traing and testing
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=0)
```

Navie Bayes

In [13]:
```
#Creating navie bayes model
text_analysis_model = MultinomialNB().fit(X_train,y_train)
```

In [14]:
```
# prediction
y_pred = text_analysis_model.predict(X_test)
```

In [15]:
```
# getting accuracy score for test data and predicted data
score = accuracy_score(y_test,y_pred)
print(score)
print(cr(y_test,y_pred))
```

```
0.9856424982053122
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1208
           1       0.95      0.94      0.95       185

    accuracy                           0.99      1393
   macro avg       0.97      0.97      0.97      1393
weighted avg       0.99      0.99      0.99      1393
```

Implementation of Lemmatization and removing stop words in the messages

In [16]:
```
lemmatizer=WordNetLemmatizer()
```

In [17]:
```
corpus = []
for i in range(0, len(spam_text)):
    messages = re.sub('[^a-zA-Z]', ' ', spam_text['Message'][i])
    messages = messages.lower()
    messages = messages.split()

    messages = [lemmatizer.lemmatize(word) for word in messages if not word in stopwords
    messages = ' '.join(messages)
    corpus.append(messages)
```

Applying BagOfWords (BOW)

In [18]:
```
# Creating Bag of Words model
cv = CountVectorizer(max_features=2500)
x = cv.fit_transform(corpus).toarray()
```

In [19]:
```
# Converting the categorical values into dummy variables
y = pd.get_dummies(spam_text['Category'])
y = y.iloc[:,1].values
```

In [20]:
```
# spliting the data for traing and testing
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=0)
```

Naive Bayes

In [21]:
```
text_analysis_model2 = MultinomialNB().fit(X_train,y_train)
```

In [22]:
```
y_prediction = text_analysis_model2.predict(X_test)
```

In [23]:
```
# getting accuracy score for test data and predicted data
score = accuracy_score(y_test,y_prediction)
print(score)
print(cr(y_test,y_prediction))
```

```
0.9834888729361091
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1208
           1       0.95      0.92      0.94       185

    accuracy                           0.98      1393
   macro avg       0.97      0.96      0.96      1393
weighted avg       0.98      0.98      0.98      1393
```

Implementing Tf-Idf for Lemmatization

In [24]:
```
# Creating a TFIDF model
from sklearn.feature_extraction.text import TfidfVectorizer
tv = TfidfVectorizer(max_features=2500)
X = tv.fit_transform(corpus).toarray()
```

In [25]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state
```

Naive bayes for Tf-Idf

In [26]:
```
text_analysis_model2 = MultinomialNB().fit(X_train,y_train)
```

In [27]:
```
y_predtf = text_analysis_model2.predict(X_test)
```

In [28]:
```
score=accuracy_score(y_test,y_predtf)
print(score)
print(cr(y_test,y_predtf))
```

```
0.97847533632287
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       955
           1       0.99      0.86      0.92       160

    accuracy                           0.98      1115
   macro avg       0.98      0.93      0.95      1115
weighted avg       0.98      0.98      0.98      1115
```

Results of Count Vectorizer (CV)

Accuracy

Stemming Naive-Bayes - 98.56% ~ (99%)

Lemmatization Naive-Bayes - 98.34% ~ (98%)

Results of TF-IDF

Lemmatization Naive-Bayes - 97.84% ~ (98%)

In [ ]: