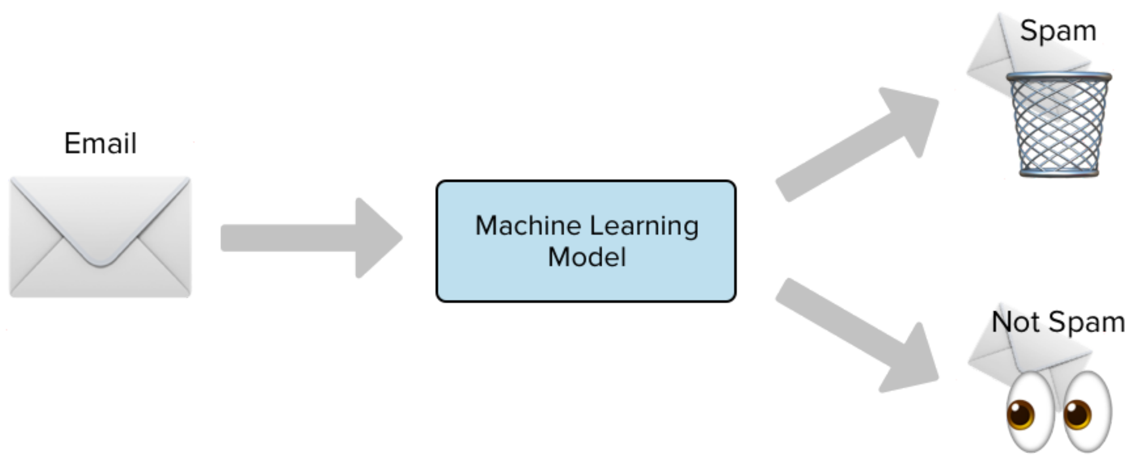
# Email Spam Detection Project

## Introduction

In this project I will be making a machine learning model in order to detect spam mails from that of the normal ones.  I will be using a dataset with pre defined labels (i.e either the email is spam or a ham) and I will be making a predictive model to differentiate  between the two.

## Problem Statement

Let's start by defining the problem that we are going to solve in this project. You are probably familiar with what spam emails are already; spam email filtering is an essential feature for email services such as Gmail, Yahoo Mail, and Outlook. Spam emails can be annoying for users, but they bring more issues and risks with them. For example, a spam email can be designed to solicit credit card numbers or bank account information, which can be used for credit card fraud or money laundering. A spam email can also be used to obtain personal data, such as a social security number or user IDs and passwords, which then can be used for identity theft and various other crimes.



## Importing Libraries

```python
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

- Above are the basic libraries that I will be using in this project. The other specialized libraries will be imported at the later stage of the project.

## Calling the Dataset

```
#reading dataset
ds_email=pd.read_csv("email.csv",encoding="latin-1")

ds_email
```

| | subject | message | label |
|---|---|---|---|
| 0 | job posting - apple-iss research center | content - length : 3386 apple-iss research cen... | 0 |
| 1 | NaN | lang classification grimes , joseph e . and ba... | 0 |
| 2 | query : letter frequencies for text identifica... | i am posting this inquiry for sergei atamas ( ... | 0 |
| 3 | risk | a colleague and i are researching the differin... | 0 |
| 4 | request book information | earlier this morning i was on the phone with a... | 0 |

- The dataset has three columns namely the subject, message and label.

## Describing Information

```
ds_email.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2893 entries, 0 to 2892
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   subject  2831 non-null   object
 1   message  2893 non-null   object
 2   label    2893 non-null   int64 
dtypes: int64(1), object(2)
memory usage: 67.9+ KB
```

- The above table shows there are total of three columns in the dataset
- One column subject has null values which later will be replaced by mode or most frequent occurring values.
- Two columns types are object and one is integer.

## Replacing Missing Values

```python
#replacing null values in the subject column with the most frequent
for col in ["subject"]:

    ds_email[col].fillna(ds_email[col].mode()[0],inplace=True)
```

- The missing values in the column subject is replaced by mode or the most frequent occurring.
- I used mode as the data type in the column is categorical

## Checking for Missing Values

```python
#checking missing values
ds_email.isnull().sum()
```

```
subject    0
message    0
label      0
dtype: int64
```

- The  above table shows that there are no missing values in the dataset as it is already replaced by mode strategy.

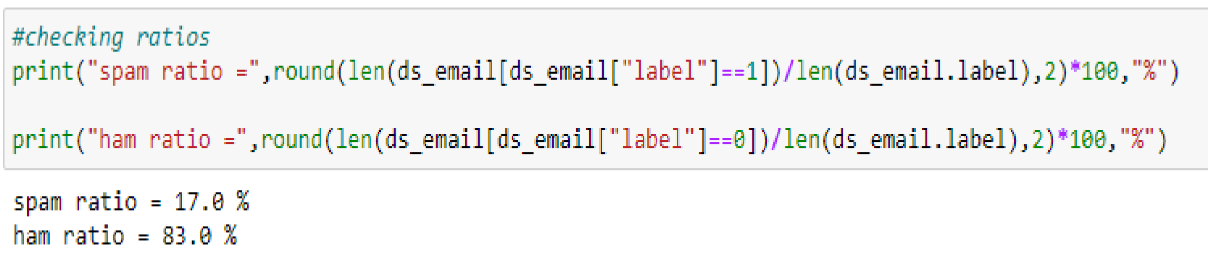
## Checking value counts

```python
#checking value counts
ds_email.label.value_counts()
```

```
0    2412
1     481
Name: label, dtype: int64
```

- The above code shows the value counts for the column label which is basically our target variable.
- We can see there are 2412 records for the 0s i.e. the Ham mails and 481 records for the 1s that is the Spam mails.

## Checking Ratios

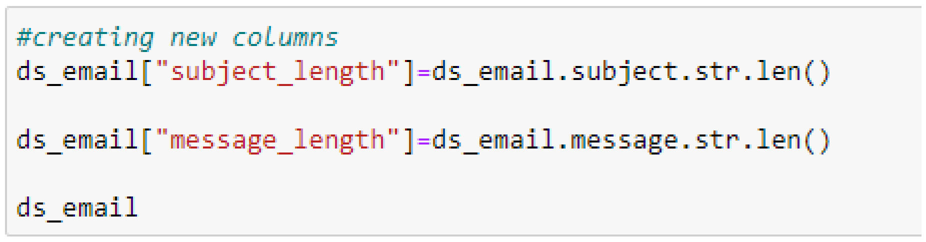


```
#checking ratios
print("spam ratio =",round(len(ds_email[ds_email["label"]==1])/len(ds_email.label),2)*100,"%")

print("ham ratio =",round(len(ds_email[ds_email["label"]==0])/len(ds_email.label),2)*100,"%")
```

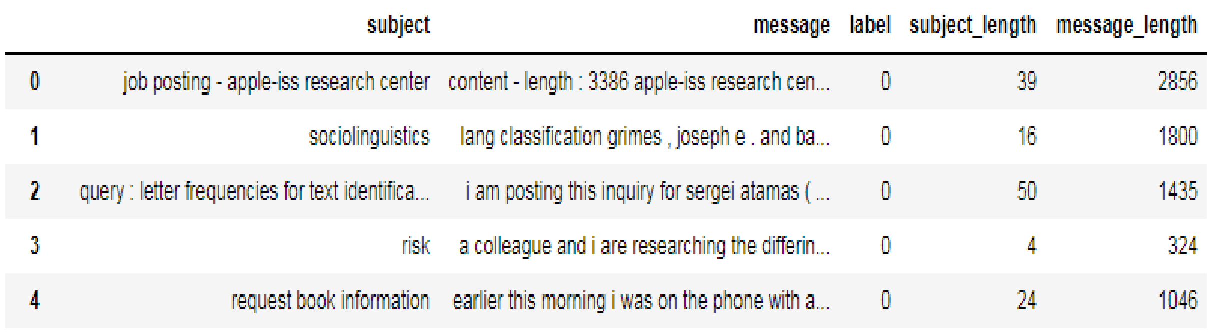
```
spam ratio = 17.0 %
ham ratio = 83.0 %
```

- The above code shows that the percentage of spam mails are just 17% where as the % of normal mails are 83%.
- This concludes that we have an imbalanced dataset which we need to take care of otherwise it will create an over fitted model.
- Later we will fixed this issues by some strategies.

## Creating New Columns



```
#creating new columns
ds_email["subject_length"]=ds_email.subject.str.len()

ds_email["message_length"]=ds_email.message.str.len()

ds_email
```



| | subject | message | label | subject_length | message_length |
|---|---|---|---|---|---|
| **0** | job posting - apple-iss research center | content - length : 3386 apple-iss research cen... | 0 | 39 | 2856 |
| **1** | sociolinguistics | lang classification grimes , joseph e . and ba... | 0 | 16 | 1800 |
| **2** | query : letter frequencies for text identifica... | i am posting this inquiry for sergei atamas ( ... | 0 | 50 | 1435 |
| **3** | risk | a colleague and i are researching the differin... | 0 | 4 | 324 |
| **4** | request book information | earlier this morning i was on the phone with a... | 0 | 24 | 1046 |

- We can see that two new columns Subject  Length and Message Length are created.
- This will help us to get an idea if the length of the messages might differentiates the spam from that of the ham emails.

## Feature Engineering

In this step we will be using techniques which will help reduce the size the data so that it is easy for the model to read it. We will be performing the below steps for the columns "message" and "subject".

```
#replacing email address with email
ds_email["message"]=ds_email["message"].str.replace(r"^.+@[^\.].*\.[a-z]{2,}$","emailaddress")
```

```
#replacing email address with email
ds_email["subject"]=ds_email["subject"].str.replace(r"^.+@[^\.].*\.[a-z]{2,}$","emailaddress")
```

```
#replace urls with webadress
ds_email["message"]=ds_email["message"].str.replace(r"^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$","webaddress")
```

```
#replace urls with webadress
ds_email["subject"]=ds_email["subject"].str.replace(r"^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$","webaddress")
```

```
#replacing money with money symbol
ds_email["message"]=ds_email["message"].str.replace(r"$","dollers")
```

```
#replacing money with money symbol
ds_email["subject"]=ds_email["subject"].str.replace(r"$","dollers")
```

- The above code shows that components of an email address replaced by the word email address.
- Components of an web address is replaced by the word web address.
- $ or any other signs are replaced by the word dollars.

```
#replacing 10 digit phone number
ds_email["message"]=ds_email["message"].str.replace(r"^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$","phonemumber")
```

```
#replacing 10 digit phone number
ds_email["subject"]=ds_email["subject"].str.replace(r"^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$","phonemumber")
```

```
#replace number with numbr
ds_email["message"]=ds_email["message"].str.replace(r"\d+(\.\d+)?","numbr")
```

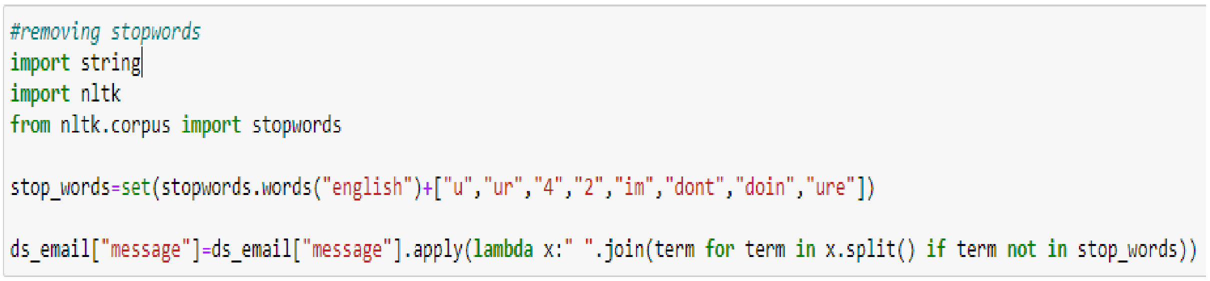
```
#replace number with numbr
ds_email["subject"]=ds_email["subject"].str.replace(r"\d+(\.\d+)?","numbr")
```

```
#replacing punctuation
ds_email["message"]=ds_email["message"].str.replace(r"[^\w\d\s]"," ")
```

```
#replacing punctuation
ds_email["subject"]=ds_email["subject"].str.replace(r"[^\w\d\s]"," ")
```

- The above code shows a phone number replaced by the word phone number.
- Punctuations replaced by blank space.
- Any number replaced by the word number.

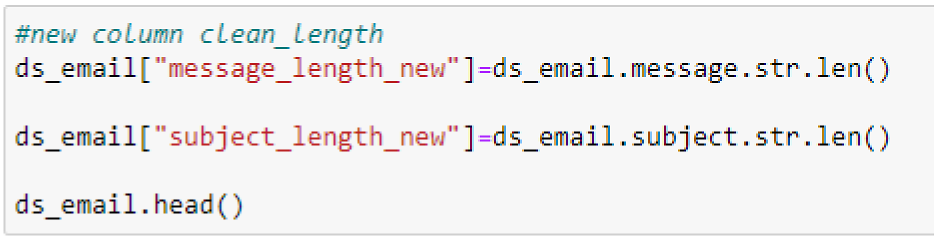
## Removing StopWords



```
#removing stopwords
import string
import nltk
from nltk.corpus import stopwords

stop_words=set(stopwords.words("english")+["u","ur","4","2","im","dont","doin","ure"])

ds_email["message"]=ds_email["message"].apply(lambda x:" ".join(term for term in x.split() if term not in stop_words))
```

- The above code is to remove the stop words from the data.
- Stop words consist of a huge part of any data and those needs to be removed as taking them will reduce the efficiency of the models.
- Also I added couple of more words which I felt might be useful  for this particular study.

## Making New Columns



```
#new column clean_length
ds_email["message_length_new"]=ds_email.message.str.len()

ds_email["subject_length_new"]=ds_email.subject.str.len()

ds_email.head()
```

- Two more columns are added to check the length of the new cleaned messages.
- This will also act as a compare to how much data is cleaned after performing the above steps.

## Displaying Length Removal

```
#total length removal
print("Message Original Length",ds_email.message_length.sum())

print("Message Clean Length",ds_email.message_length_new.sum())

print("Subject Original Length",ds_email.subject_length.sum())

print("Subject Clean Length",ds_email.subject_length_new.sum())
```

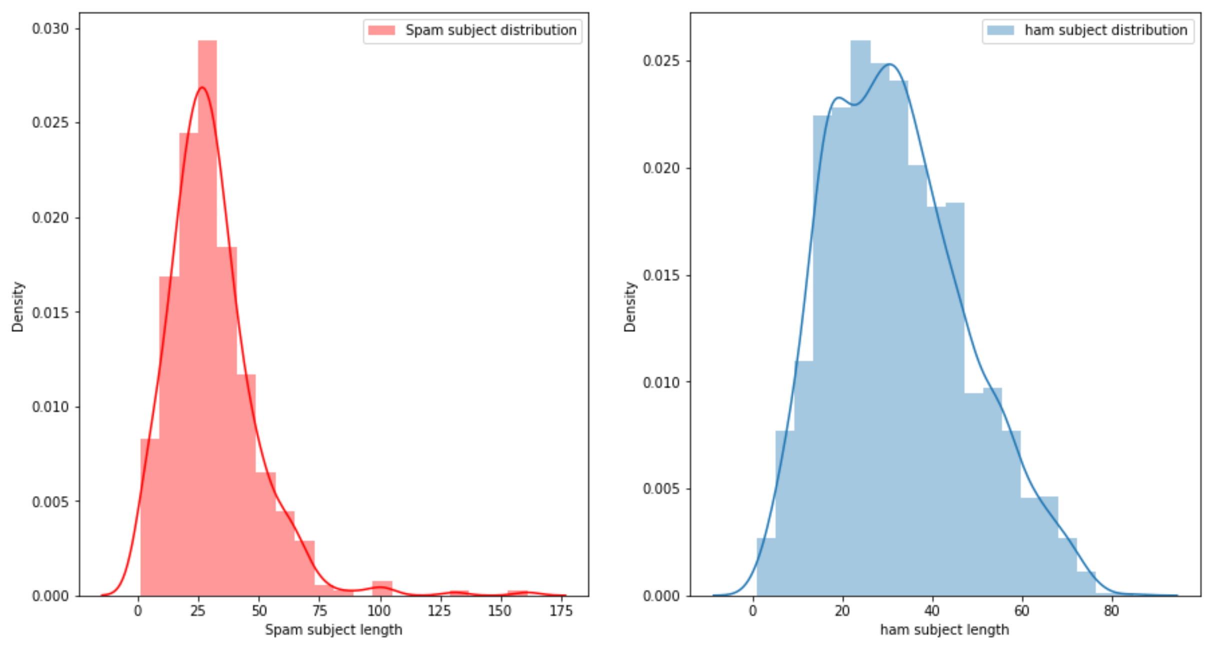
```
Message Original Length 9070005
Message Clean Length 6568010
Subject Original Length 92639
Subject Clean Length 89187
```

- From the above code we can see that there is a huge amount of characters cleaned after performing the feature engineering in the dataset.

## Plotting Distribution

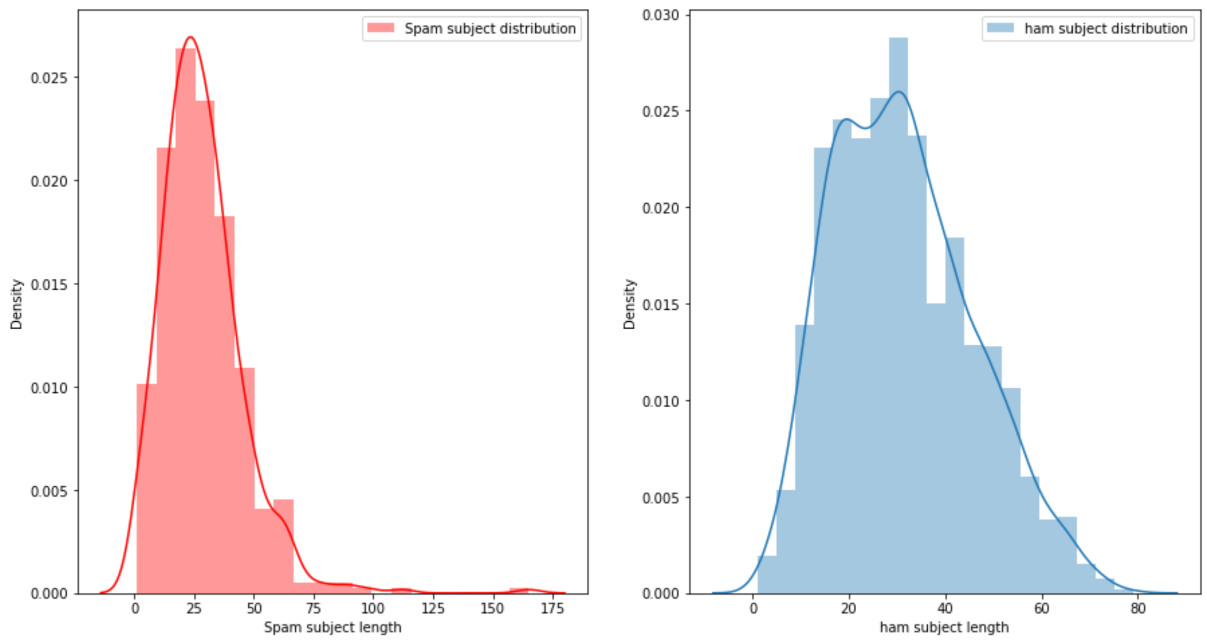
In this step we will be plotting the distribution of the length of normal text to that of the cleaned texts. This will help us to have a visual understanding of what differentiates normal mails from that of spam mails.

```
#plotting distribution of original subject length of words
f,ax=plt.subplots(1,2,figsize=(15,8))

sns.distplot(ds_email[ds_email["label"]==1]["subject_length"],bins=20,ax=ax[0],label="Spam subject distribution",color="r")

ax[0].set_xlabel("Spam subject length")

ax[0].legend()

sns.distplot(ds_email[ds_email["label"]==0]["subject_length"],bins=20,ax=ax[1],label="ham subject distribution")

ax[1].set_xlabel("ham subject length")

ax[1].legend()

plt.show()
```



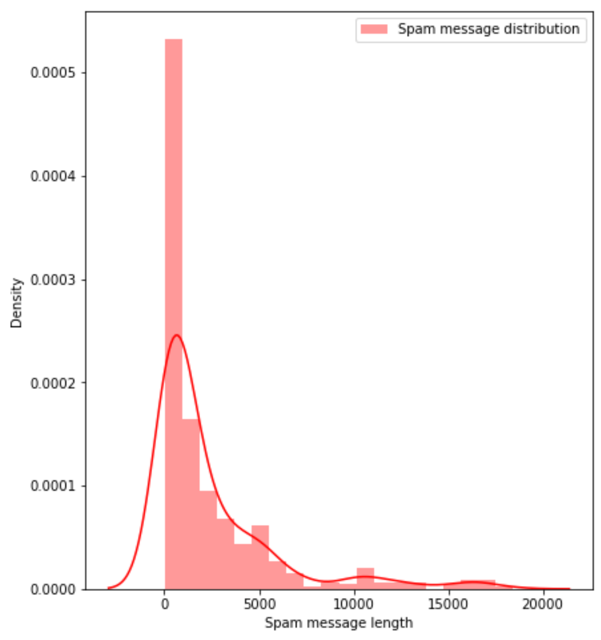
- The above are the non cleaned distribution of length of the subject column.
- We can see that normal mails subjects ranges from 0 to 100 letters where as spam mails subjects are from 0 to 175 letters.

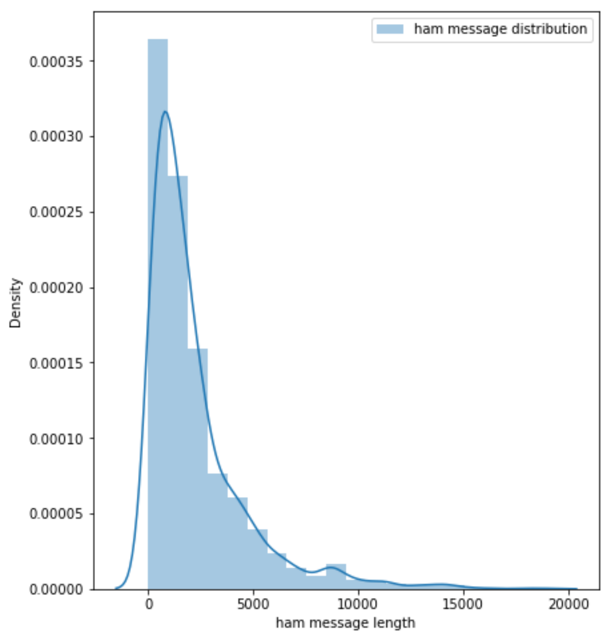
```
#plotting distribution of cleaned subject length of words
f,ax=plt.subplots(1,2,figsize=(15,8))

sns.distplot(ds_email[ds_email["label"]==1]["subject_length_new"],bins=20,ax=ax[0],label="Spam subject distribution",color="r")

ax[0].set_xlabel("Spam subject length")

ax[0].legend()

sns.distplot(ds_email[ds_email["label"]==0]["subject_length_new"],bins=20,ax=ax[1],label="ham subject distribution")

ax[1].set_xlabel("ham subject length")

ax[1].legend()

plt.show()
```



- The above are the cleaned distribution of length of the subject column.
- We can see that normal mails subject length ranges from 0 to 70 letters where as spam mails messages are from 0 to 175 letters.
-

```
#plotting distribution of original message length of words
f,ax=plt.subplots(1,2,figsize=(15,8))

sns.distplot(ds_email[ds_email["label"]==1]["message_length_new"],bins=20,ax=ax[0],label="Spam message distribution",color="r")

ax[0].set_xlabel("Spam message length")

ax[0].legend()

sns.distplot(ds_email[ds_email["label"]==0]["message_length_new"],bins=20,ax=ax[1],label="ham message distribution")

ax[1].set_xlabel("ham message length")

ax[1].legend()

plt.show()
```

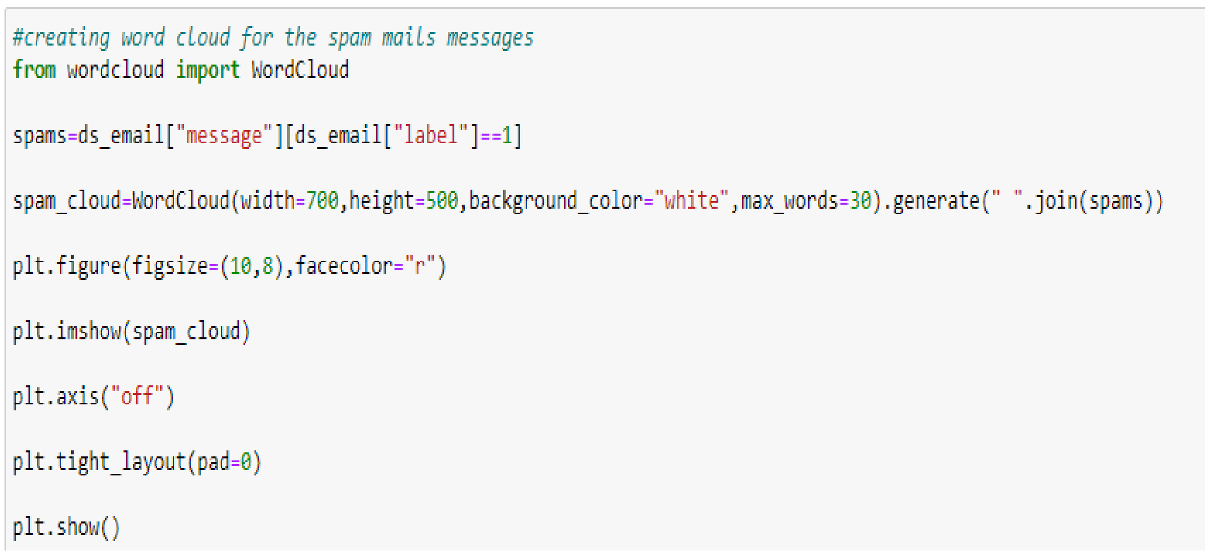




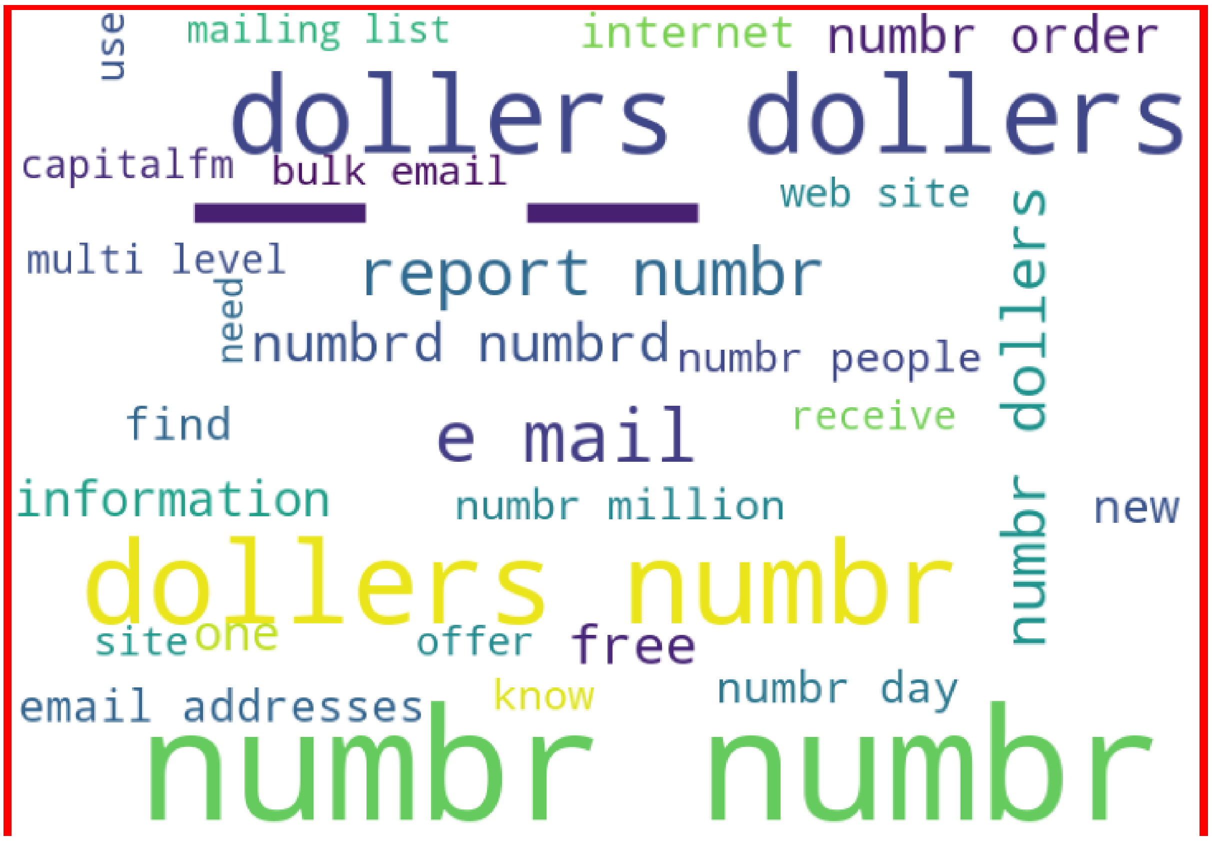
- The above are the cleaned distribution of length of the message column
- We can see that normal mails message length ranges from 0 to 2000 letters whereas spam mails subjects are from 0 to 2500 letters.

## Creating WordCloud

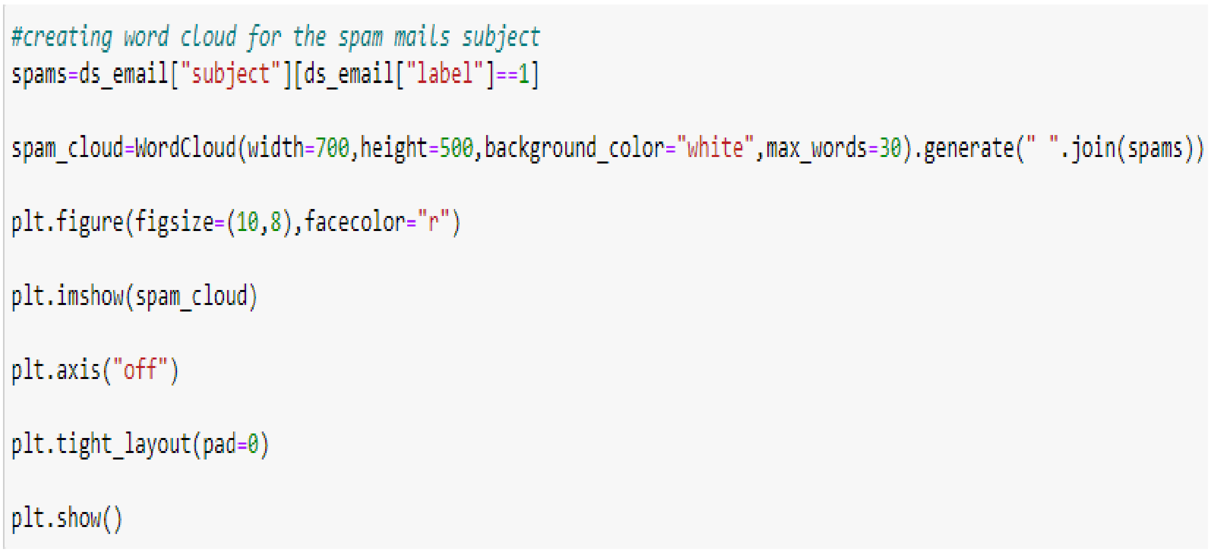
Creating a Word Cloud will help us to visualize the most repeating words in the dataset and will help us to understand if we are going in the right direction in our study.



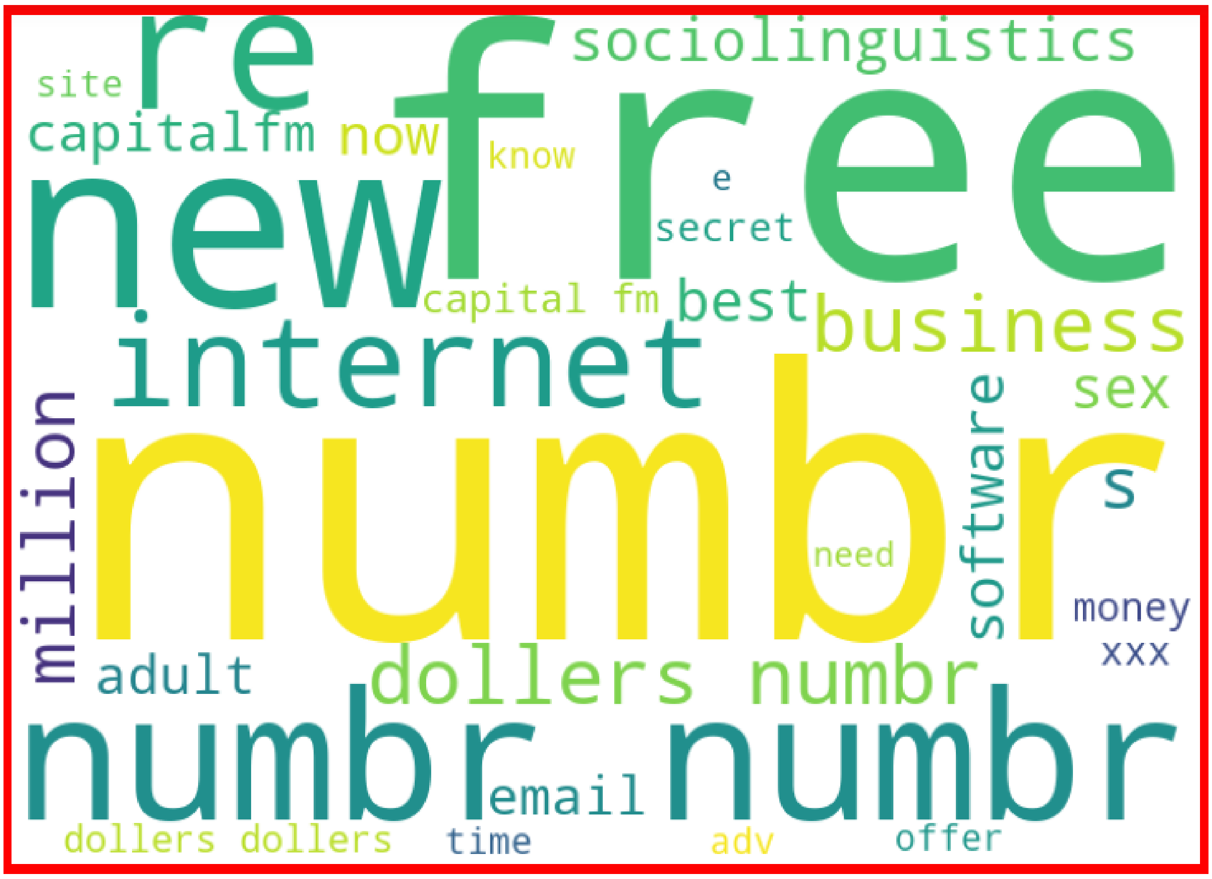
```python
#creating word cloud for the spam mails messages
from wordcloud import WordCloud

spams=ds_email["message"][ds_email["label"]==1]

spam_cloud=WordCloud(width=700,height=500,background_color="white",max_words=30).generate(" ".join(spams))

plt.figure(figsize=(10,8),facecolor="r")

plt.imshow(spam_cloud)

plt.axis("off")

plt.tight_layout(pad=0)

plt.show()
```



- The above WordCloud shows the most repeating words in the message column for the spam emails.

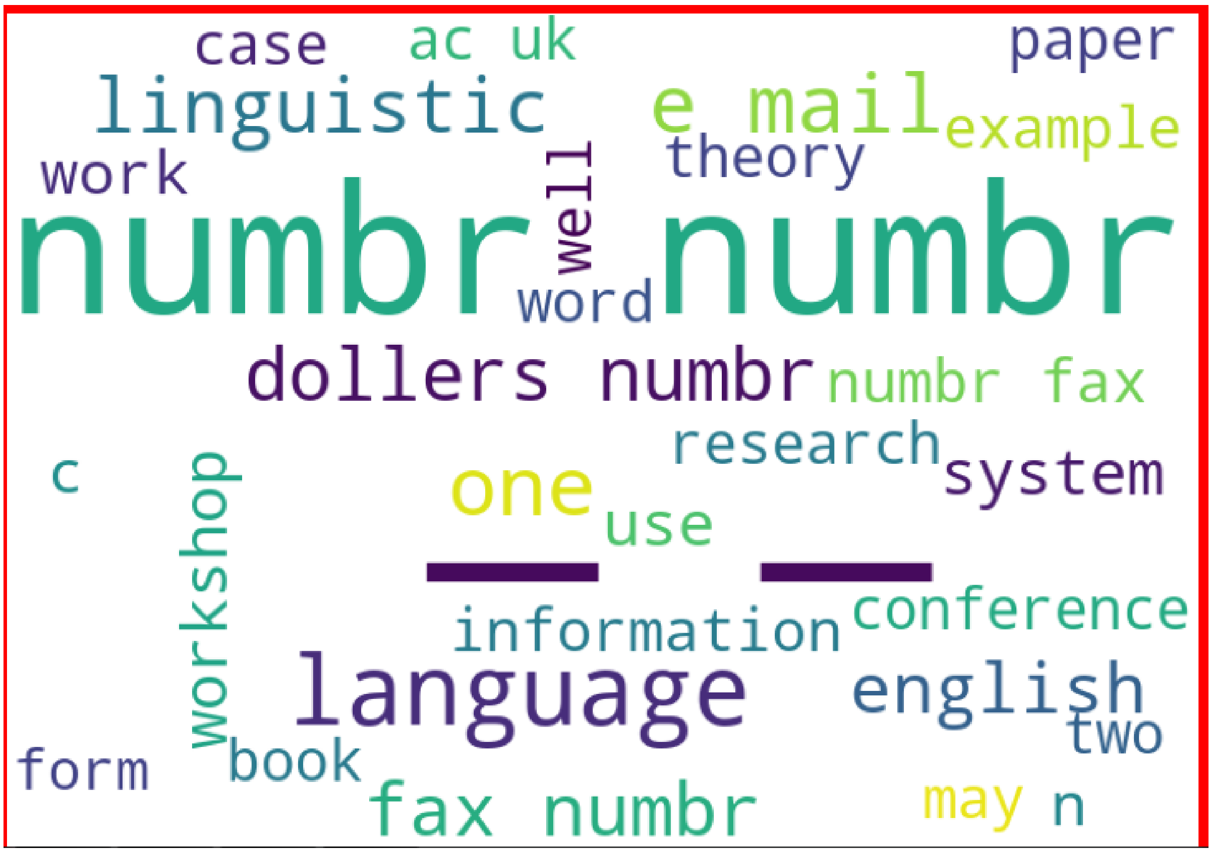


```
#creating word cloud for the spam mails subject
spams=ds_email["subject"][ds_email["label"]==1]

spam_cloud=WordCloud(width=700,height=500,background_color="white",max_words=30).generate(" ".join(spams))

plt.figure(figsize=(10,8),facecolor="r")

plt.imshow(spam_cloud)

plt.axis("off")

plt.tight_layout(pad=0)

plt.show()
```



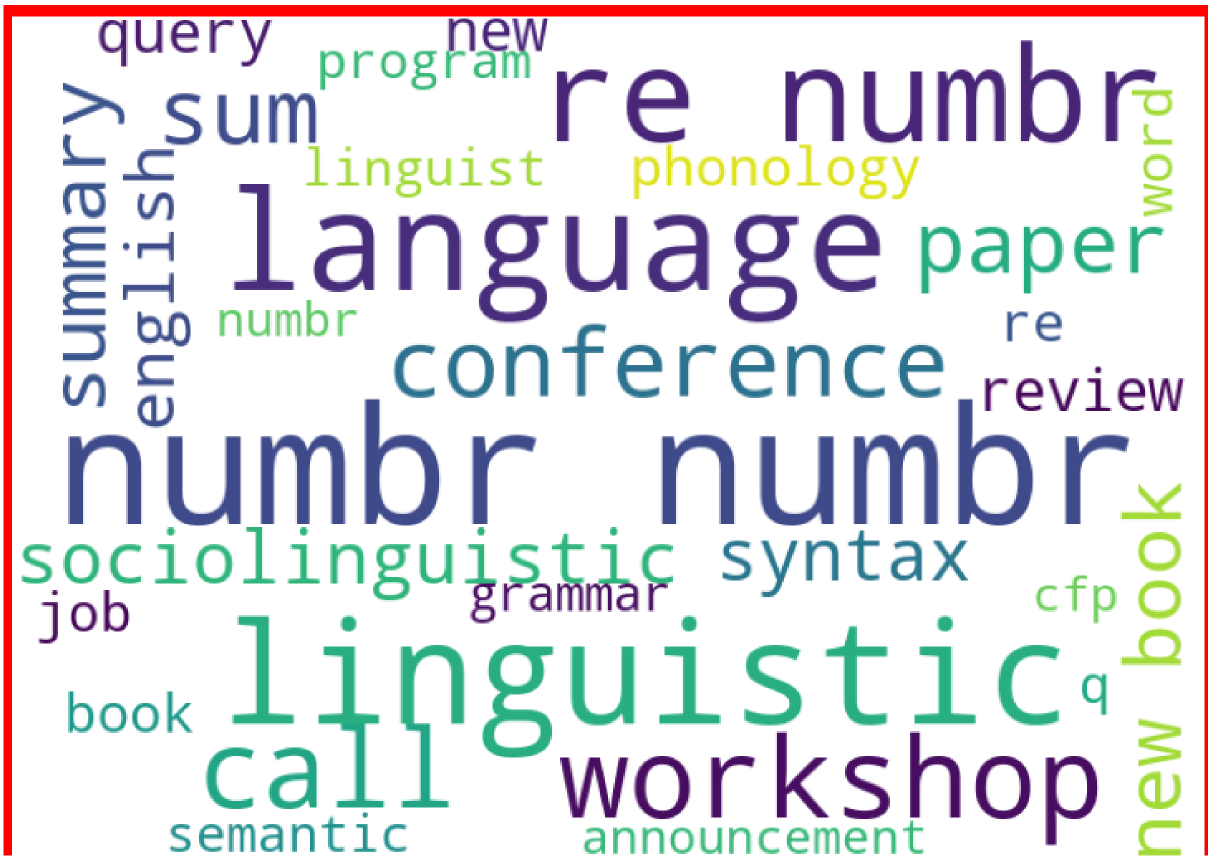
- The above WordCloud shows the most repeating words in the subject column for the spam emails.

```
#creating word cloud for the ham mails messages
from wordcloud import WordCloud

hams=ds_email["message"][ds_email["label"]==0]

ham_cloud=WordCloud(width=700,height=500,background_color="white",max_words=30).generate(" ".join(hams))

plt.figure(figsize=(10,8),facecolor="r")

plt.imshow(ham_cloud)

plt.axis("off")

plt.tight_layout(pad=0)

plt.show()
```



- The above WordCloud shows the most repeating words in the message column for the ham emails.

```
#creating word cloud for the ham mails subject
from wordcloud import WordCloud

hams=ds_email["subject"][ds_email["label"]==0]

ham_cloud=WordCloud(width=700,height=500,background_color="white",max_words=30).generate(" ".join(hams))

plt.figure(figsize=(10,8),facecolor="r")

plt.imshow(ham_cloud)

plt.axis("off")

plt.tight_layout(pad=0)

plt.show()
```



- The above WordCloud shows the most repeating words in the subject column for the ham emails.

**Importing Algorithms and Metrics**

```
#important algorithims and metrices
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

- Above are the algorithms which will be used for training the model and also evaluating the model based on various parameters.
- We will be using Naive Bayes algorithm as it works best with text classifications.

**Splitting Target and Input Variable**

```
tf_vec=TfidfVectorizer()

naive=MultinomialNB()

features=tf_vec.fit_transform(ds_email["message"],ds_email["subject"])

x=features

y=ds_email["label"]
```

- The above code splits the dataset into Target that is the Dependent variable and Input that is the Independent variable.

**Splitting Training and Testing Data**

```
#traning and predicting
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2169, 54356)
(724, 54356)
(2169,)
(724,)
```

- The above code split the entire dataset into training and testing variables.
- We  can also see the shape of the training and testing dataset.

## Over Sampling using SMOTE Algorithm

```
#importing SMOTE algorithim
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state = 2)

x_train_res, y_train_res = sm.fit_resample(x_train, y_train)
```

```
#counting before and after over sampling
from collections import Counter

counter=Counter(y_train)

print("Before",counter)

counter=Counter(y_train_res)

print("After",counter)
```

```
Before Counter({0: 1827, 1: 342})
After Counter({0: 1827, 1: 1827})
```

- Before we have checked and confirmed that we have class imbalances in this dataset.
- So we will have used the SMOTE algorithm to over sample the minority class.

## Accuracy Score and Cross Val Score

```
naive.fit(x_train_res,y_train_res)

y_pred=naive.predict(x_test)

print("Accuracy_Score =",accuracy_score(y_test,y_pred))

print("\n")

cross_val= cross_val_score(naive,x,y,cv=10,scoring="accuracy").mean()

print("Cross_val_score=",cross_val)
```

```
Accuracy_Score = 0.9834254143646409


Cross_val_score= 0.8589631308913017
```

From the above code we can see that the accuracy score of the model is around 98% and cross validation score is   85%.

## Classification Report and ROC AUC score

```
#plotting classification report
false_positive_rate,true_positive_rate,thresholds=roc_curve(y_test,y_pred)

roc_auc=auc(false_positive_rate,true_positive_rate)

print("roc_auc_score=",roc_auc)

print("\n")

print("classification_report\n",classification_report(y_test,y_pred))
```

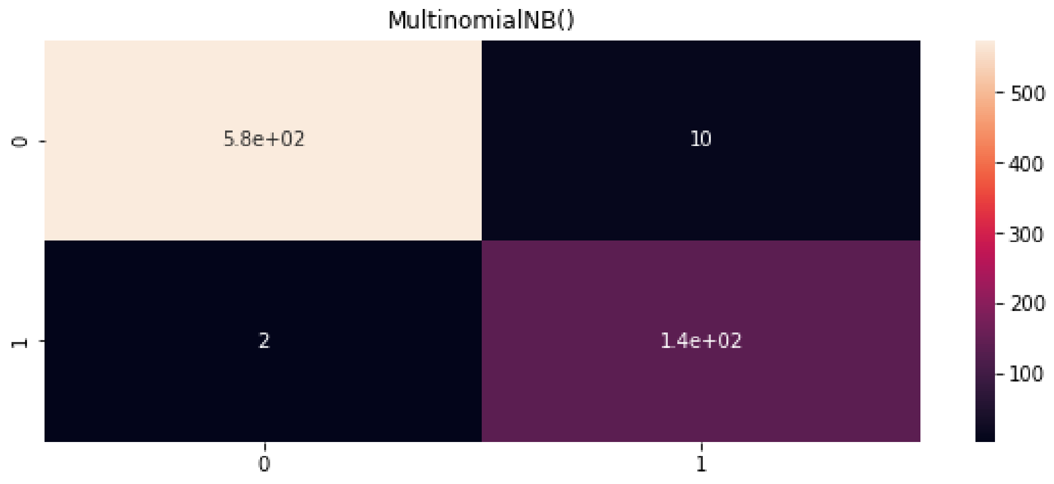
```
roc_auc_score= 0.9842587468486749


classification_report
               precision    recall  f1-score   support

           0       1.00      0.98      0.99       585
           1       0.93      0.99      0.96       139

    accuracy                           0.98       724
   macro avg       0.96      0.98      0.97       724
weighted avg       0.98      0.98      0.98       724
```

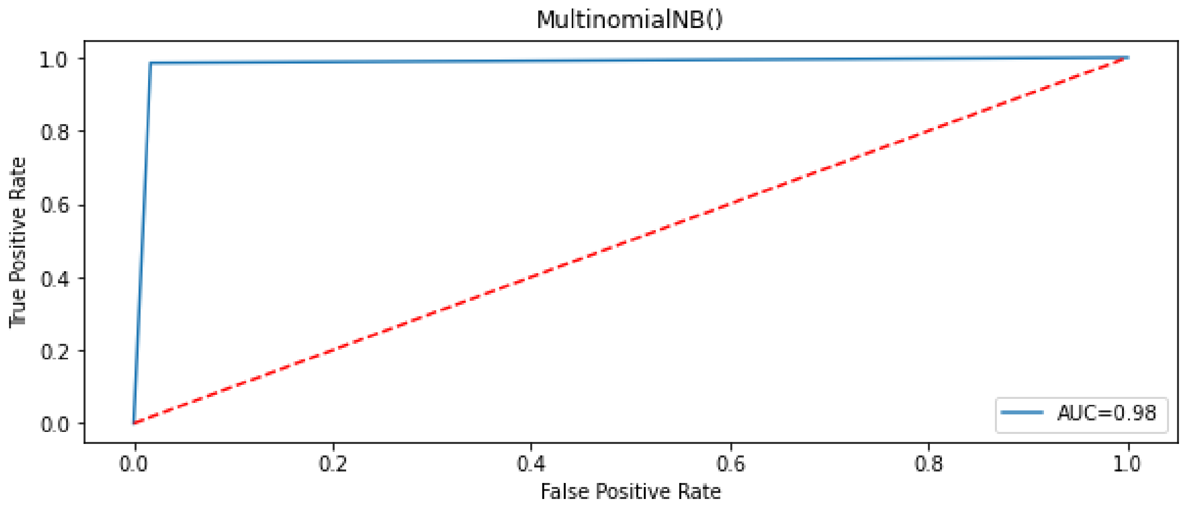
- The ROC AUC score for the model is 98% and also the classification report is listed in the top.

## Confusion Matrix and ROC AUC curve

The below code is to plot the confusion matrix and ROC AUC curve

```
#plotting confusion matrix
cm=confusion_matrix(y_test,y_pred)

print(cm)

print("\n")

plt.figure(figsize=(10,40))

plt.subplot(911)

plt.title(naive)

print(sns.heatmap(cm,annot=True))

plt.subplot(912)

plt.title(naive)

plt.plot(false_positive_rate,true_positive_rate,label="AUC=%0.2f"% roc_auc)

plt.plot([0,1],[0,1],"r--")

plt.legend(loc="lower right")

plt.ylabel("True Positive Rate")

plt.xlabel("False Positive Rate")

print("\n\n")
```


MultinomialNB()
5.8e+02
1.4e+02


MultinomialNB()
True Positive Rate
False Positive Rate
AUC=0.98