Project Report on

EMAIL SPAM DETECTION

FLIP ROBO

NAME OF THE PROJECT

EMAIL SPAM DETECTION

Submitted by:

Mr. Vikas Kumar Mishra

FLIPROBO SME:

Ms Khushboo Garg

Project Report on

EMAIL SPAM DETECTION

ACKNOWLEDGMENT

I would like to express my special gratitude to the “Flip Robo” team, who has allowed me to deal with a beautiful dataset and helped me improve my analysis skills. And I want to express my huge gratitude to Ms Khushboo Garg (SME Flip Robo).
Thanks to “Data trained” who are the reason behind my Internship at FlipRobo Technologies.

SOURCE USED IN THIS PROJECT:

1. Learn Library Documentation
2. Help from YouTube Channels, Blogs from Educational Websites
3. Notes on Machine Learning (YouTube Channel)
4. SCIKIT Learn Library Documentation
5. Help from Kaggle websites, analytical vidya, GeeksforGeeks, etc.

Project Report on

EMAIL SPAM DETECTION

INTRODUCTION

What is a spam filtering?

Spam Detector is used to detect unwanted, malicious and virus-infected texts and helps to separate them from non-spam texts. It uses a binary type of classification containing the labels such as **‘ham’** (non-spam) and **spam**. This application can be seen in Google Mail (GMAIL) where it segregates spam emails to prevent them from getting into the user’s inbox.

Context

The SMS Spam Collection is a set of SMS-tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according to being ham (legitimate) or spam.

Project Report on

EMAIL SPAM DETECTION

Workflow

In this project, we are using some machine learning and Natural language processing libraries like NLTK, re (Regular Expression), Scikit Learn

Natural Language Processing

Machine learning data only works with numerical features so we have to convert text data into numerical columns. So, we have to pre-process the text, called natural language processing

In-text pre-processing, we clean our text by steaming, lemmatization, removing stop words, removing special symbols and numbers, etc. After cleaning the data, we have to feed this text data into a vectorizer which will convert this text data into numerical features.

Dataset

The files contain one message per line. Each line is composed of two columns: v1 contains the label (ham or spam) and v2 contains the raw text.

Project Report on

EMAIL SPAM DETECTION

This corpus has been collected from free or free research sources on the Internet:

A collection of 5573 rows of SMS spam messages was manually extracted from the Grumble text Web site. This is a UK forum in which cell phone users make public claims about SMS spam messages, most of them without reporting the very spam message received. The identification of the text of spam messages in the claims is a very hard and time-consuming task, and it involved carefully scanning hundreds of web pages.

A subset of 3,375 SMS randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore. The messages largely originate from Singaporeans and mostly from students attending the University. These messages were collected from volunteers who were made aware that their contributions were going to be made publicly available.

Analytical Problem Framing

Mathematical / Analytical Modelling of the Problem

Project Report on

EMAIL SPAM DETECTION

Our objective is to detect spam messages which can be resolved by the use of the classification-based algorithm. In this project, we will use different types of algorithms that use their mathematical equation in the background. This project comes with a single data set for spam files. Initially, data cleaning & pre-processing perform over data. Feature engineering is performed to remove unnecessary features & for dimensionality reduction.
we are using some machine learning and Natural language processing libraries like NLTK, re (Regular Expression), Scikit Learn
In model building, the Final model is selected based on evaluation benchmarks among different models with different algorithms.

Data Sources and their formats

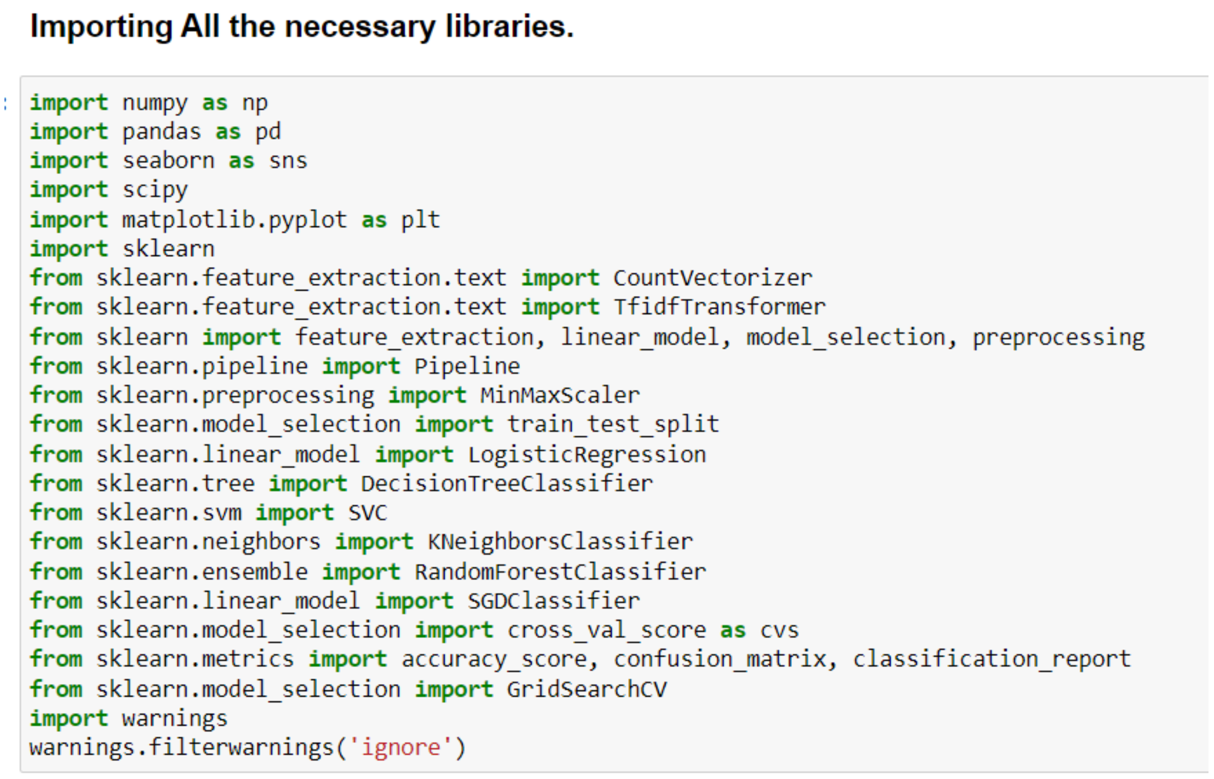
The data set provided by Flip Robo was in the format of CSV (Comma Separated Values). There is 1 data set that is given.

- There is one dataset spam.
- v1 contains the label (ham or spam) and v2 contains the raw text.
- A collection of 5573 rows of SMS spam messages was manually extracted from the Grumble text Web site.

Project Report on

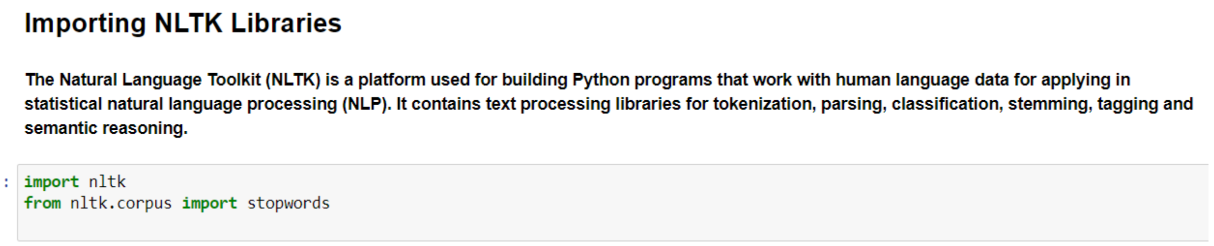
EMAIL SPAM DETECTION

First Import Libraries



Importing All the necessary libraries.

```
import numpy as np
import pandas as pd
import seaborn as sns
import scipy
import matplotlib.pyplot as plt
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn import feature_extraction, linear_model, model_selection, preprocessing
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import cross_val_score as cvs
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
```



Importing NLTK Libraries

The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.

```
import nltk
from nltk.corpus import stopwords
```

Project Report on

EMAIL SPAM DETECTION

Reading and Understanding the Data

Spam Data

```
df_spam = pd.read_csv('spam.csv', encoding='latin-1')
```

```
df_spam.head(10)
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN
5	spam	FreeMsg Hey there darling it's been 3 week's n...	NaN	NaN	NaN
6	ham	Even my brother is not like to speak with me. ...	NaN	NaN	NaN
7	ham	As per your request 'Melle Melle (Oru Minnamin...	NaN	NaN	NaN
8	spam	WINNER!! As a valued network customer you have...	NaN	NaN	NaN
9	spam	Had your mobile 11 months or more? U R entitle...	NaN	NaN	NaN

```
df_spam.shape
```

```
(5572, 5)
```

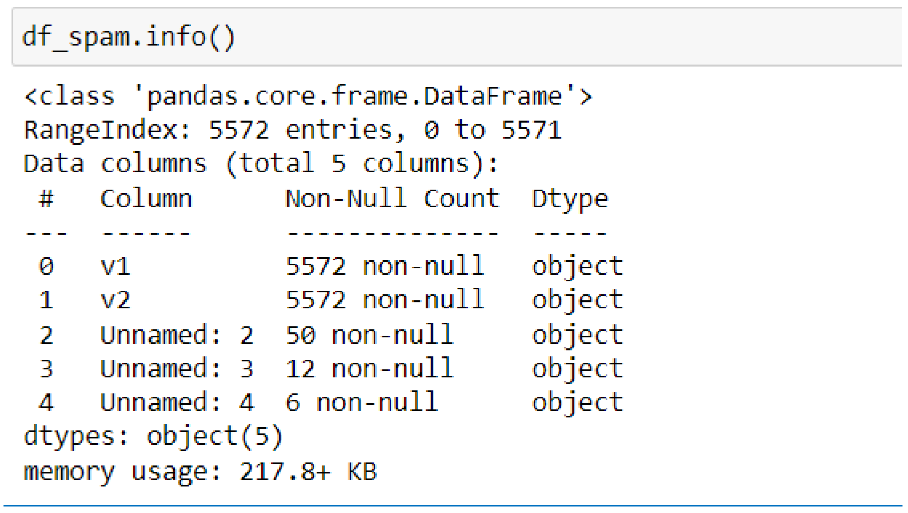
- The Spam dataset contains 5 columns with 5572 rows.

Data Cleaning and Preparation

Project Report on

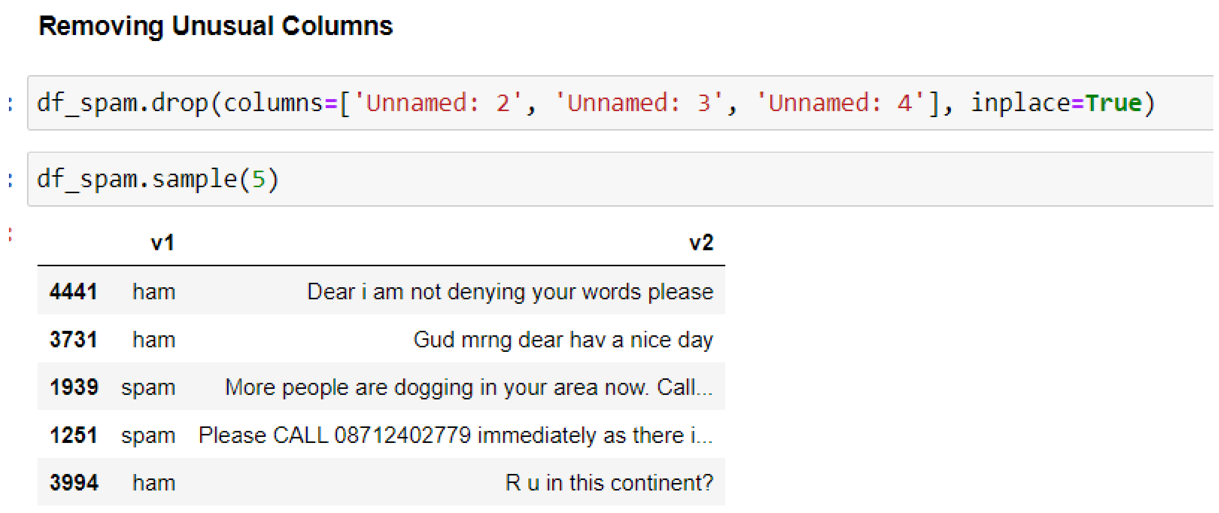
EMAIL SPAM DETECTION

1. Data Cleaning And Preparation



```
df_spam.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```



Removing Unusual Columns

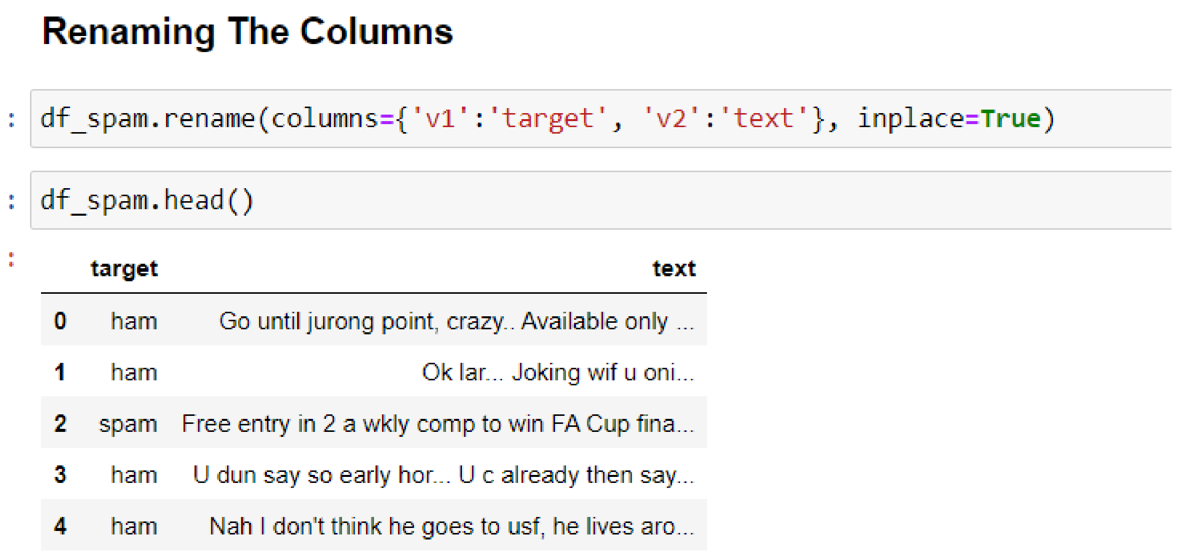
```
df_spam.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], inplace=True)
```

```
df_spam.sample(5)
```

	v1	v2
4441	ham	Dear i am not denying your words please
3731	ham	Gud mrng dear hav a nice day
1939	spam	More people are dogging in your area now. Call...
1251	spam	Please CALL 08712402779 immediately as there i...
3994	ham	R u in this continent?

Renaming The Columns

With the help of the rename method, we rename the columns name.



Renaming The Columns

```
df_spam.rename(columns={'v1':'target', 'v2':'text'}, inplace=True)
```

```
df_spam.head()
```

```
  target                                          text
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...
```

Encoding The Target Column

Encoding, at its most basic, is a method of converting characters (such as letters, punctuation, symbols, whitespace, and control characters) to numbers and, eventually, bits. Each character can be encoded to a distinct bit sequence.

Encoding The Target Column

```
from sklearn.preprocessing import LabelEncoder
```

```
encoder = LabelEncoder()
```

```
df_spam['target'] = encoder.fit_transform(df_spam['target'])
```

```
df_spam.head()
```

	target	text
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...

Project Report on

EMAIL SPAM DETECTION

Checking Missing Values and Duplicate Values

Missing Value Chek

```
df_spam.isnull().sum()
```

```
target    0
text      0
dtype: int64
```

Chek Duplicate Values

```
df_spam.duplicated().sum()
```

```
403
```

Remove Duplicates Values

```
df_spam = df_spam.drop_duplicates(keep='first')
```

```
df_spam.duplicated().sum()
```

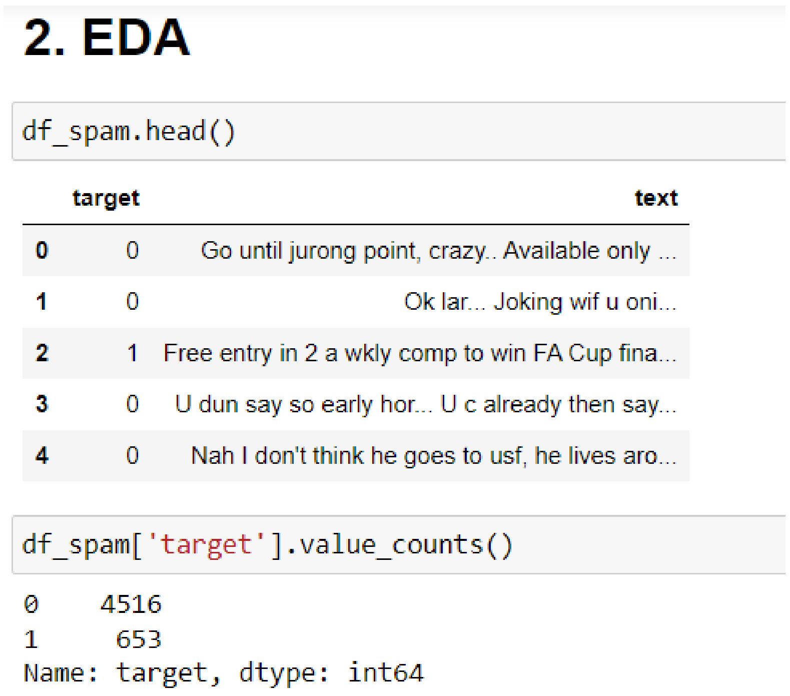
```
0
```

```
df_spam.shape
```

```
(5169, 2)
```

EDA Process

2. EDA



```
df_spam.head()
```

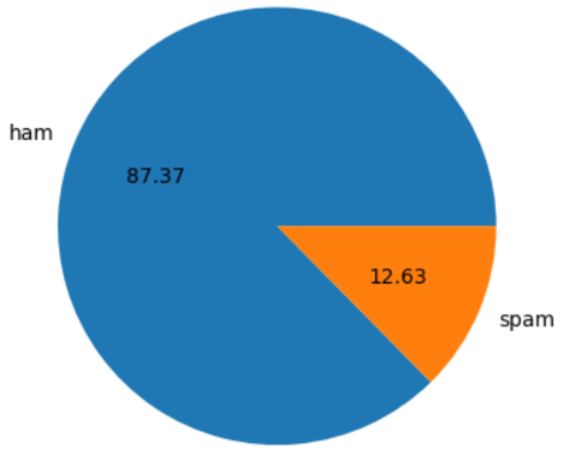
	target	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

```
df_spam['target'].value_counts()
```

```
0    4516
1     653
Name: target, dtype: int64
```

Creating Pie Chart

```
plt.pie(df_spam['target'].value_counts(), labels=['ham', 'spam'], autopct="%0.2f")
plt.show()
```



Data is Imbalanced

We can see in the pie-chart ham messeges are 87% and spam messeges are only 12.6%, so our data is imbalanced

Download Punkt

```
import nltk
```

```
nltk.download('punkt')
```

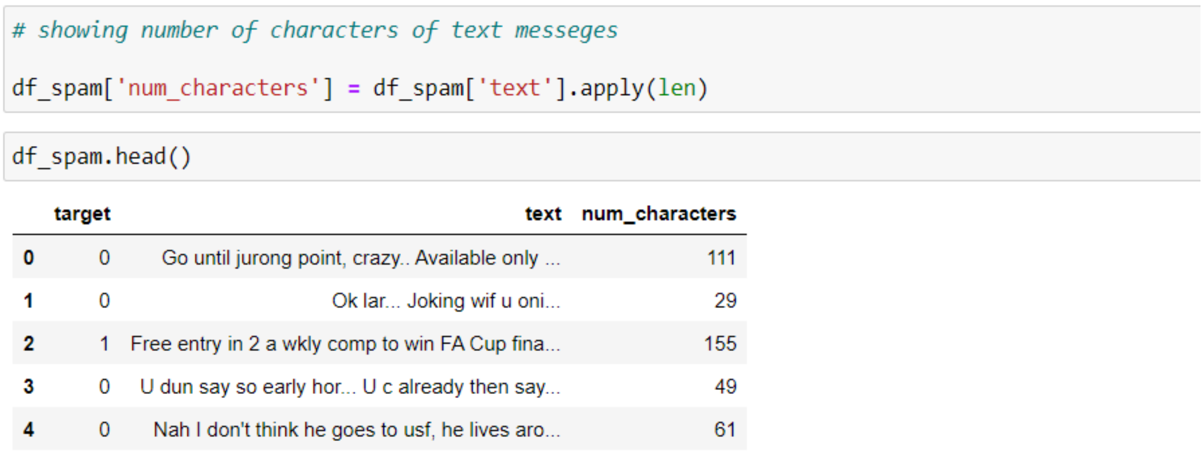
```
[nltk_data] Downloading package punkt to C:\Users\Vikash
[nltk_data]     Mishra\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
True
```

Punkt:

This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plaintext in the target language before it can be used.

Adding New Columns

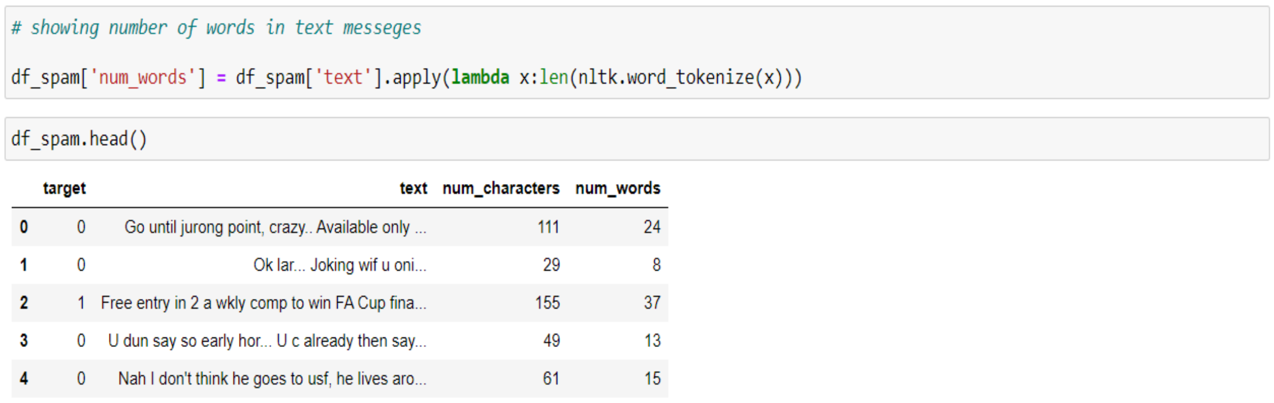


```
# showing number of characters of text messeges

df_spam['num_characters'] = df_spam['text'].apply(len)
```

```
df_spam.head()
```

	target	text	num_characters
0	0	Go until jurong point, crazy.. Available only ...	111
1	0	Ok lar... Joking wif u oni...	29
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	0	U dun say so early hor... U c already then say...	49
4	0	Nah I don't think he goes to usf, he lives aro...	61

We can add new column 'num_characters' for showing number of characters in messeges.



```
# showing number of words in text messeges

df_spam['num_words'] = df_spam['text'].apply(lambda x:len(nltk.word_tokenize(x)))
```

```
df_spam.head()
```

	target	text	num_characters	num_words
0	0	Go until jurong point, crazy.. Available only ...	111	24
1	0	Ok lar... Joking wif u oni...	29	8
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37
3	0	U dun say so early hor... U c already then say...	49	13
4	0	Nah I don't think he goes to usf, he lives aro...	61	15

We can add 2nd new column 'num_words' with the help of lambda function for tokenize the messeges words for showing number of words uses in messeges.

```
# showing numbers of sentences in text messeges

df_spam['num_sentences'] = df_spam['text'].apply(lambda x:len(nltk.sent_tokenize(x)))
```

```
df_spam.head()
```

	target	text	num_characters	num_words	num_sentences
0	0	Go until jurong point, crazy.. Available only ...	111	24	2
1	0	Ok lar... Joking wif u oni...	29	8	2
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2
3	0	U dun say so early hor... U c already then say...	49	13	1
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1

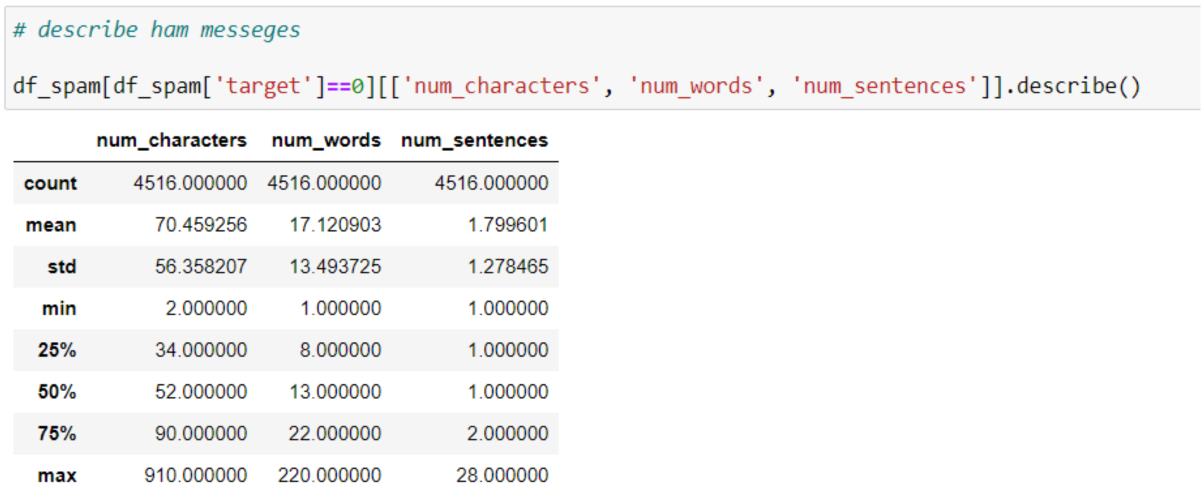
We can add 3rd new column 'num_sentences' with the help of lambda function for tokenize the sentences of messeges for showing number of sentences using in messeges.

Describe () New Columns ham and spam messages

```
df_spam[['num_characters', 'num_words', 'num_sentences']].describe()
```

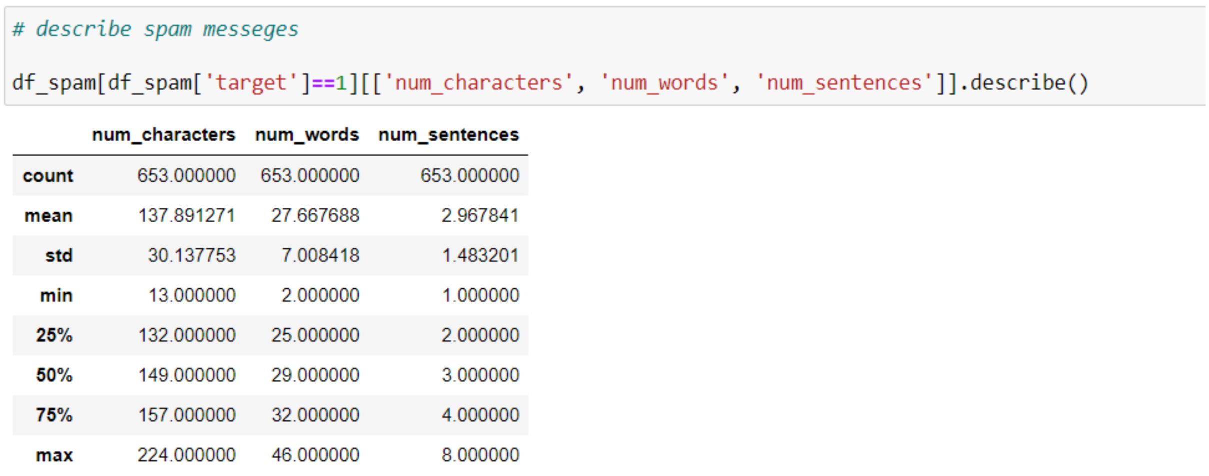
	num_characters	num_words	num_sentences
count	5169.000000	5169.000000	5169.000000
mean	78.977945	18.453279	1.947185
std	58.236293	13.324793	1.362406
min	2.000000	1.000000	1.000000
25%	36.000000	9.000000	1.000000
50%	60.000000	15.000000	1.000000
75%	117.000000	26.000000	2.000000
max	910.000000	220.000000	28.000000

Using describe() function for describing ['num_characters', 'num_words', 'num_sentences'] columns.



```
# describe ham messeges

df_spam[df_spam['target']==0][['num_characters', 'num_words', 'num_sentences']].describe()
```

	num_characters	num_words	num_sentences
count	4516.000000	4516.000000	4516.000000
mean	70.459256	17.120903	1.799601
std	56.358207	13.493725	1.278465
min	2.000000	1.000000	1.000000
25%	34.000000	8.000000	1.000000
50%	52.000000	13.000000	1.000000
75%	90.000000	22.000000	2.000000
max	910.000000	220.000000	28.000000



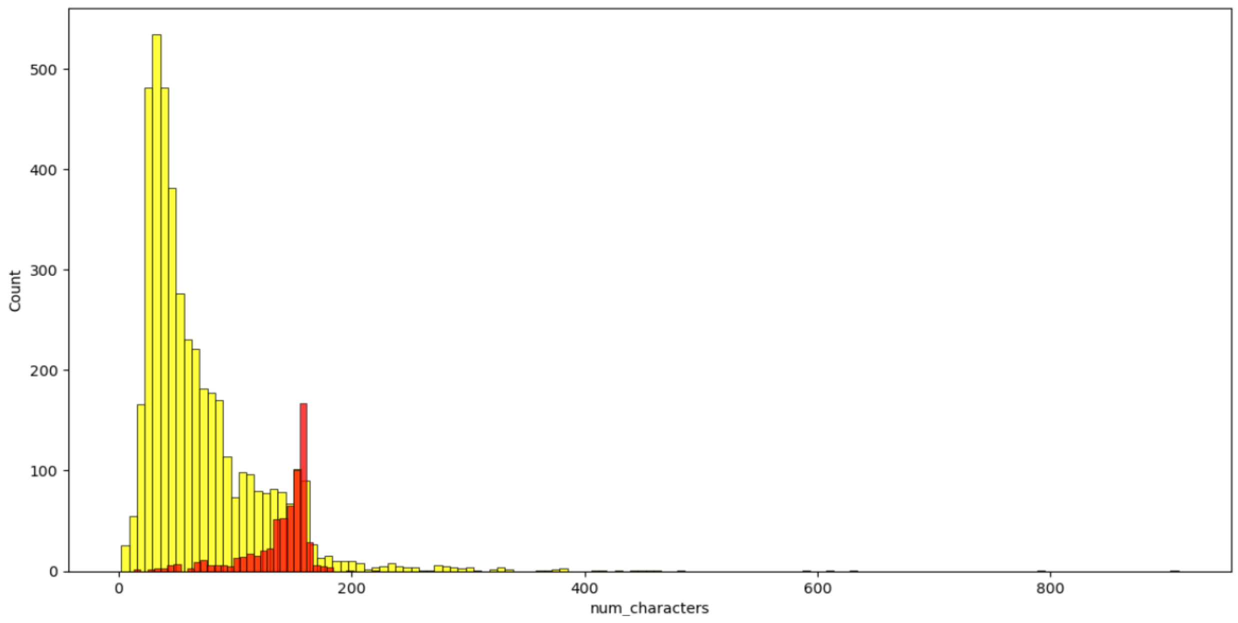
```
# describe spam messeges

df_spam[df_spam['target']==1][['num_characters', 'num_words', 'num_sentences']].describe()
```

	num_characters	num_words	num_sentences
count	653.000000	653.000000	653.000000
mean	137.891271	27.667688	2.967841
std	30.137753	7.008418	1.483201
min	13.000000	2.000000	1.000000
25%	132.000000	25.000000	2.000000
50%	149.000000	29.000000	3.000000
75%	157.000000	32.000000	4.000000
max	224.000000	46.000000	8.000000

Plotting Histogram for Ham and Spam Messages

```
plt.figure(figsize=(14,7))
sns.histplot(df_spam[df_spam['target']==0]['num_characters'], color='yellow')
sns.histplot(df_spam[df_spam['target']==1]['num_characters'], color='red')
```

```
<AxesSubplot:xlabel='num_characters', ylabel='Count'>
```



0 = ham messeges

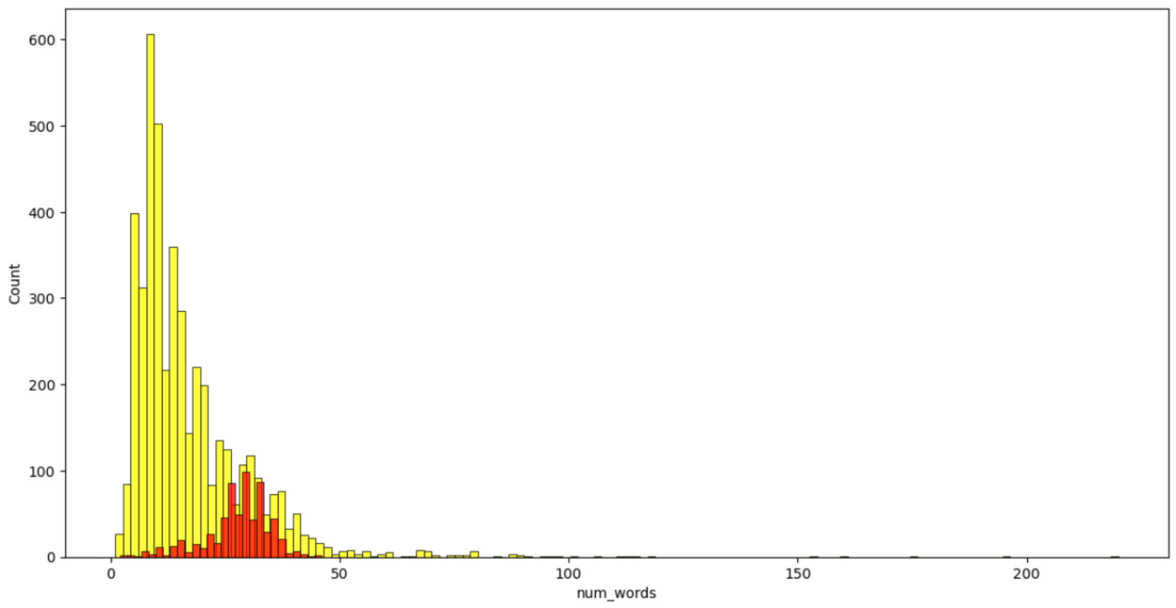
1 = spam messeges

We can see in this histogram plot yellow line for ham messeges and red line for spam messeges.

We can see number of characters of ham messeges are high.

```
plt.figure(figsize=(14,7))
sns.histplot(df_spam[df_spam['target']==0]['num_words'], color='yellow')
sns.histplot(df_spam[df_spam['target']==1]['num_words'], color='red')
```

```
<AxesSubplot:xlabel='num_words', ylabel='Count'>
```



0 = ham messeges

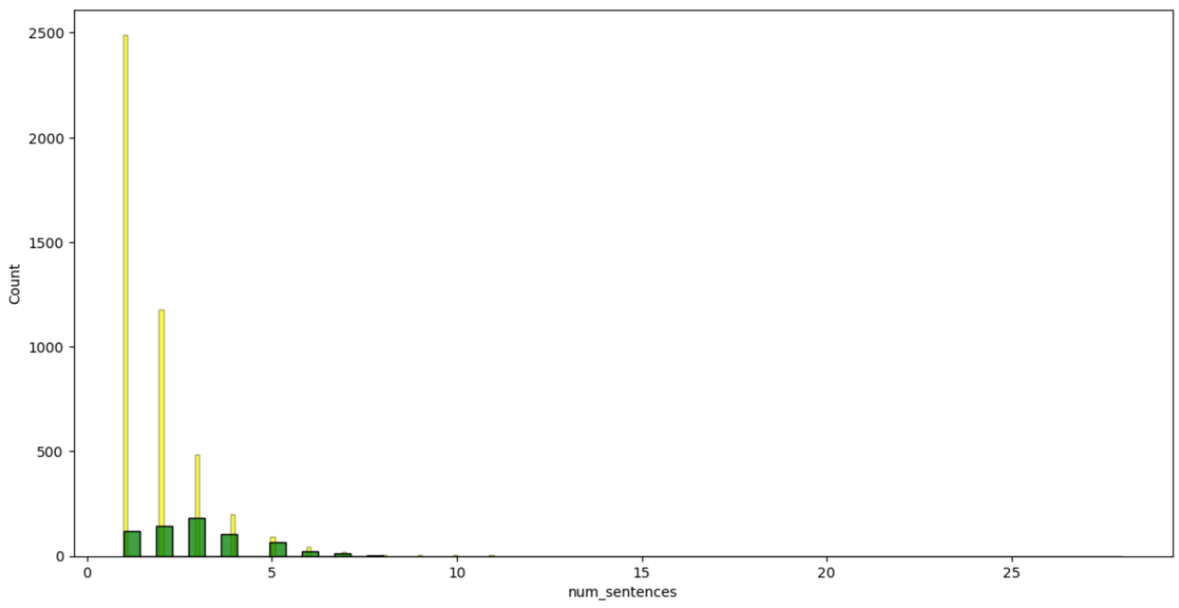
1 = spam messeges

We can see in this histogram plot yellow line for ham messeges and red line for spam messeges.

We can see number of words of ham messeges are higher than spam messeges.

```
plt.figure(figsize=(14,7))
sns.histplot(df_spam[df_spam['target']==0]['num_sentences'], color='yellow')
sns.histplot(df_spam[df_spam['target']==1]['num_sentences'], color='green')
```

```
<AxesSubplot:xlabel='num_sentences', ylabel='Count'>
```



0 = ham messeges

1 = spam messeges

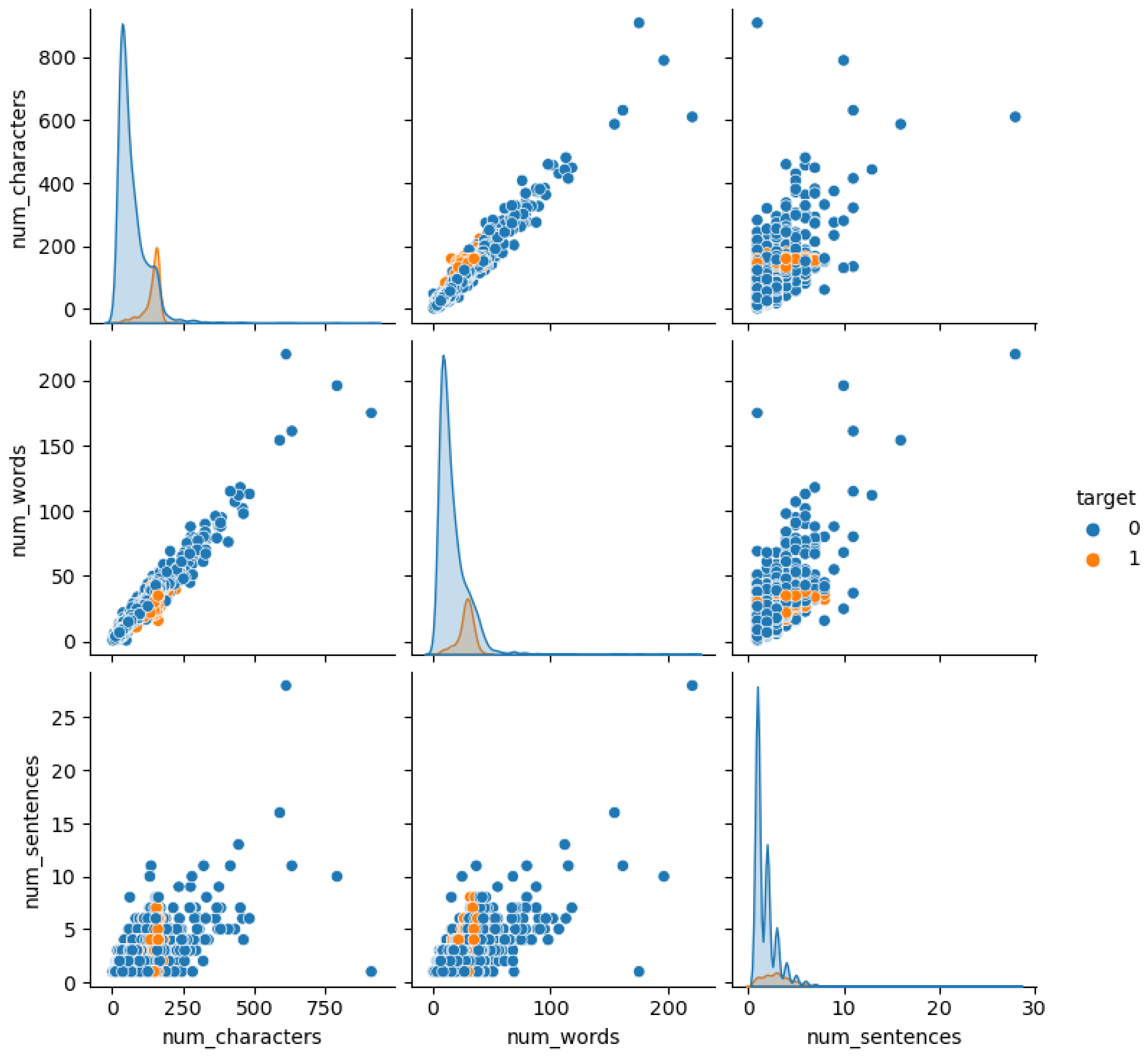
We can see in this histogram plot yellow line for ham messeges and green line for spam messeges.

We can see number of sentences of ham messeges are higher than spam messeges.

Plotting Pair Plot

```
sns.pairplot(df_spam, hue='target')
```

```
<seaborn.axisgrid.PairGrid at 0x1adf2461eb0>
```



We can see in pair plot of ham or spam messeges :

Blue for ham messeges and orange for spam messeges.

In pair plot blue dots are more than orange dots, in our data ham meseges are higher than spam messeges.

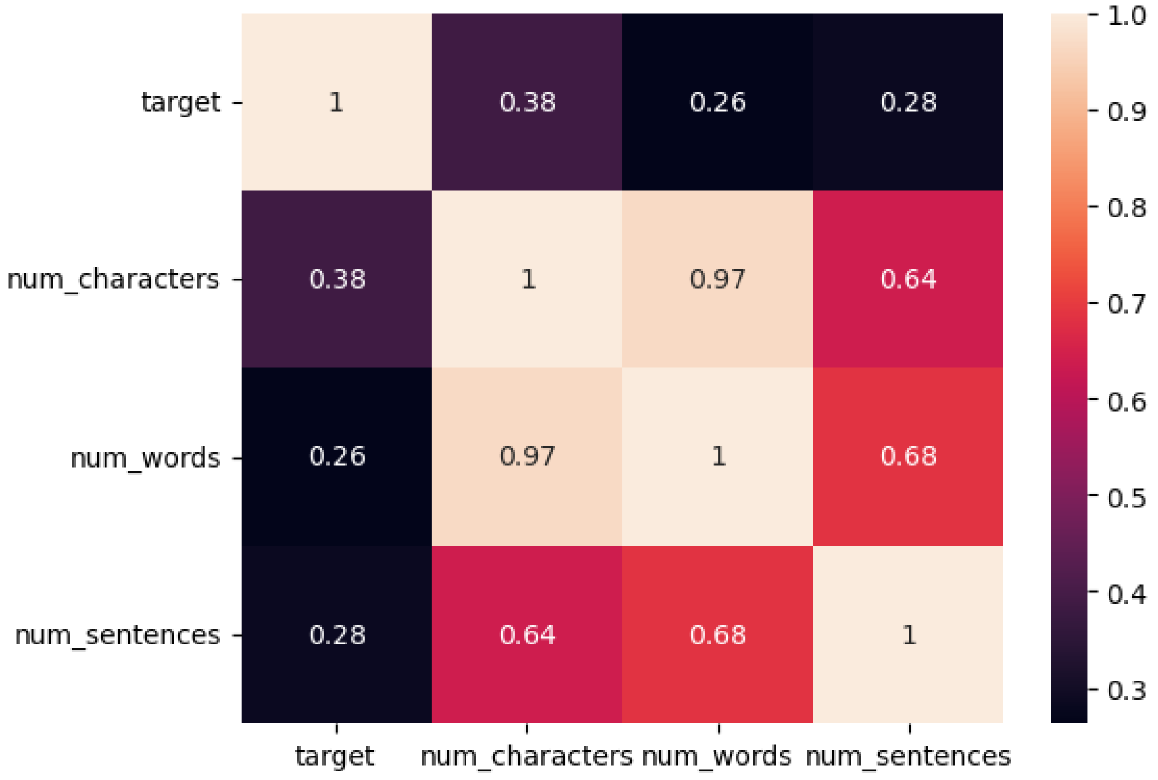
Correlation Between Columns

```
df_spam.corr()
```

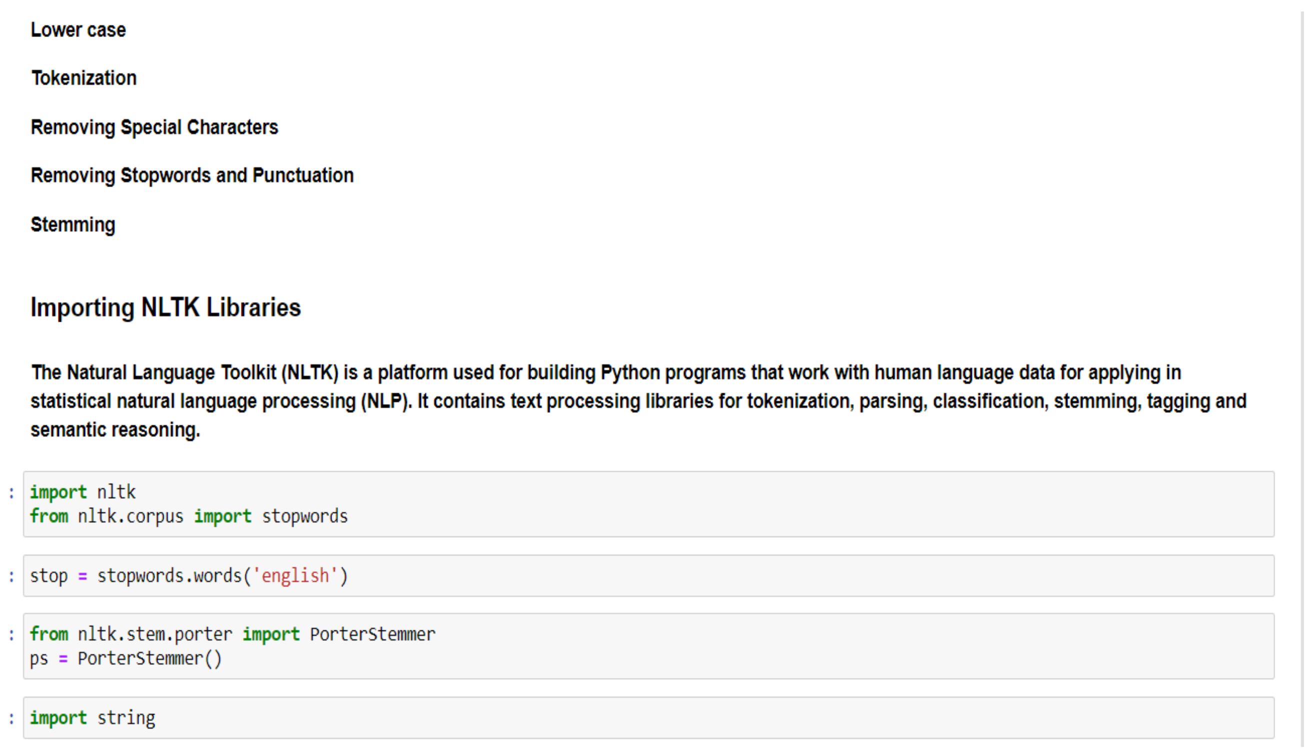
	target	num_characters	num_words	num_sentences
target	1.000000	0.384717	0.262984	0.284901
num_characters	0.384717	1.000000	0.965770	0.638143
num_words	0.262984	0.965770	1.000000	0.684541
num_sentences	0.284901	0.638143	0.684541	1.000000

```
# see correlation in heatmap

sns.heatmap(df_spam.corr(), annot=True)
```

```
<AxesSubplot:>
```



3. Data Pre-processing



Lower case

Tokenization

Removing Special Characters

Removing Stopwords and Punctuation

Stemming

Importing NLTK Libraries

The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.

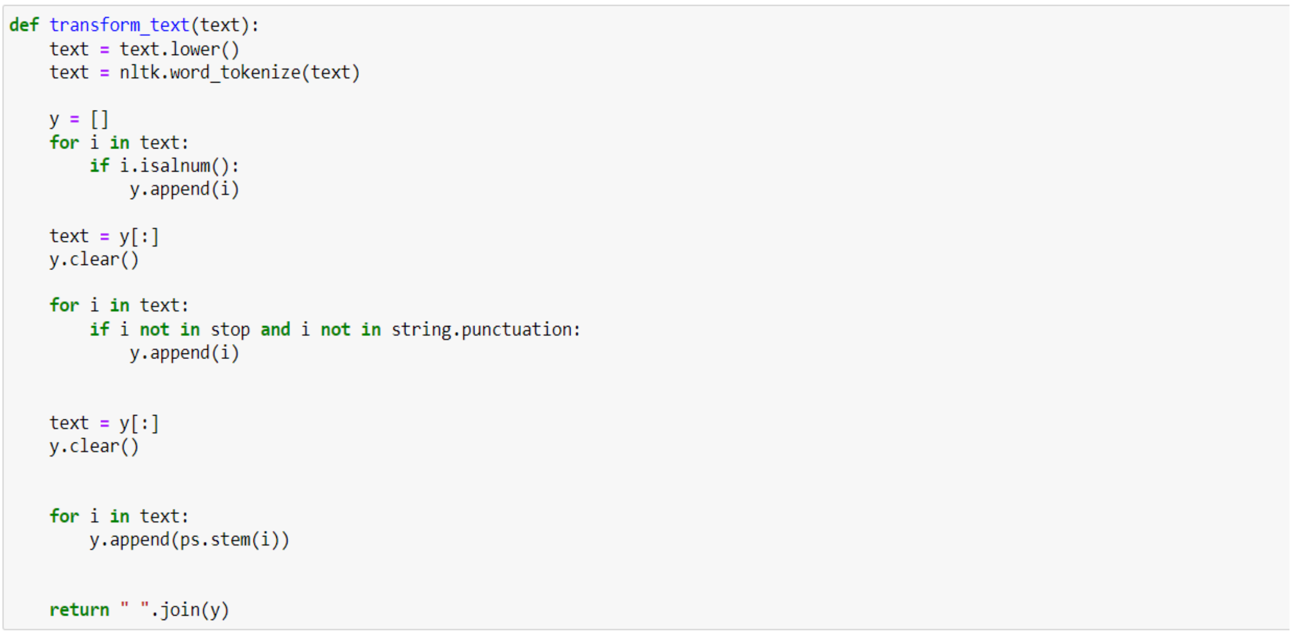
```
import nltk
from nltk.corpus import stopwords
```

```
stop = stopwords.words('english')
```

```
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
```

```
import string
```

Create Function



```
def transform_text(text):
    text = text.lower()
    text = nltk.word_tokenize(text)

    y = []
    for i in text:
        if i.isalnum():
            y.append(i)

    text = y[:]
    y.clear()

    for i in text:
        if i not in stop and i not in string.punctuation:
            y.append(i)


    text = y[:]
    y.clear()


    for i in text:
        y.append(ps.stem(i))


    return " ".join(y)
```

Creating a function (transform_text) for word tokenize, punctuation and stop words removal, lemitization, converting lowercase and stemming.

Applying Function

Convert Data into Lower Case

Lowercase letters are used for common nouns and for every letter after the initial letter of the first word of a sentence, so we convert text into lowercase.

Removing Punctuations

Punctuations are special symbols that add grammatical structure to natural English. Natural English strings are not easily processed; hence we need to remove punctuation from strings before we can use them for further processing.

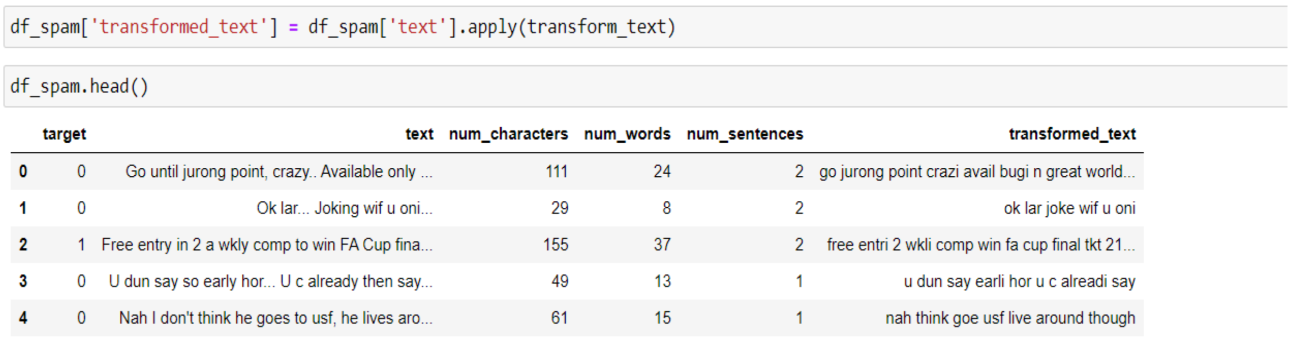
Project Report on

EMAIL SPAM DETECTION

Removing Stop Words

Stop words are available in abundance in any human language. By removing these words, we remove the low-level information from our text in order to give more focus to the important information.

Adding one more column



```
df_spam['transformed_text'] = df_spam['text'].apply(transform_text)
```

```
df_spam.head()
```

	target	text	num_characters	num_words	num_sentences	transformed_text
0	0	Go until jurong point, crazy.. Available only ...	111	24	2	go jurong point crazi avail bugi n great world...
1	0	Ok lar... Joking wif u oni...	29	8	2	ok lar joke wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2	free entri 2 wkli comp win fa cup final tkt 21...
3	0	U dun say so early hor... U c already then say...	49	13	1	u dun say earli hor u c alreadi say
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1	nah think goe usf live around though

After completing all transformation of messeges adding one more column 'transformed_text'

Word Cloud For Frequency of Images and Word

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud.

```
from wordcloud import WordCloud
```

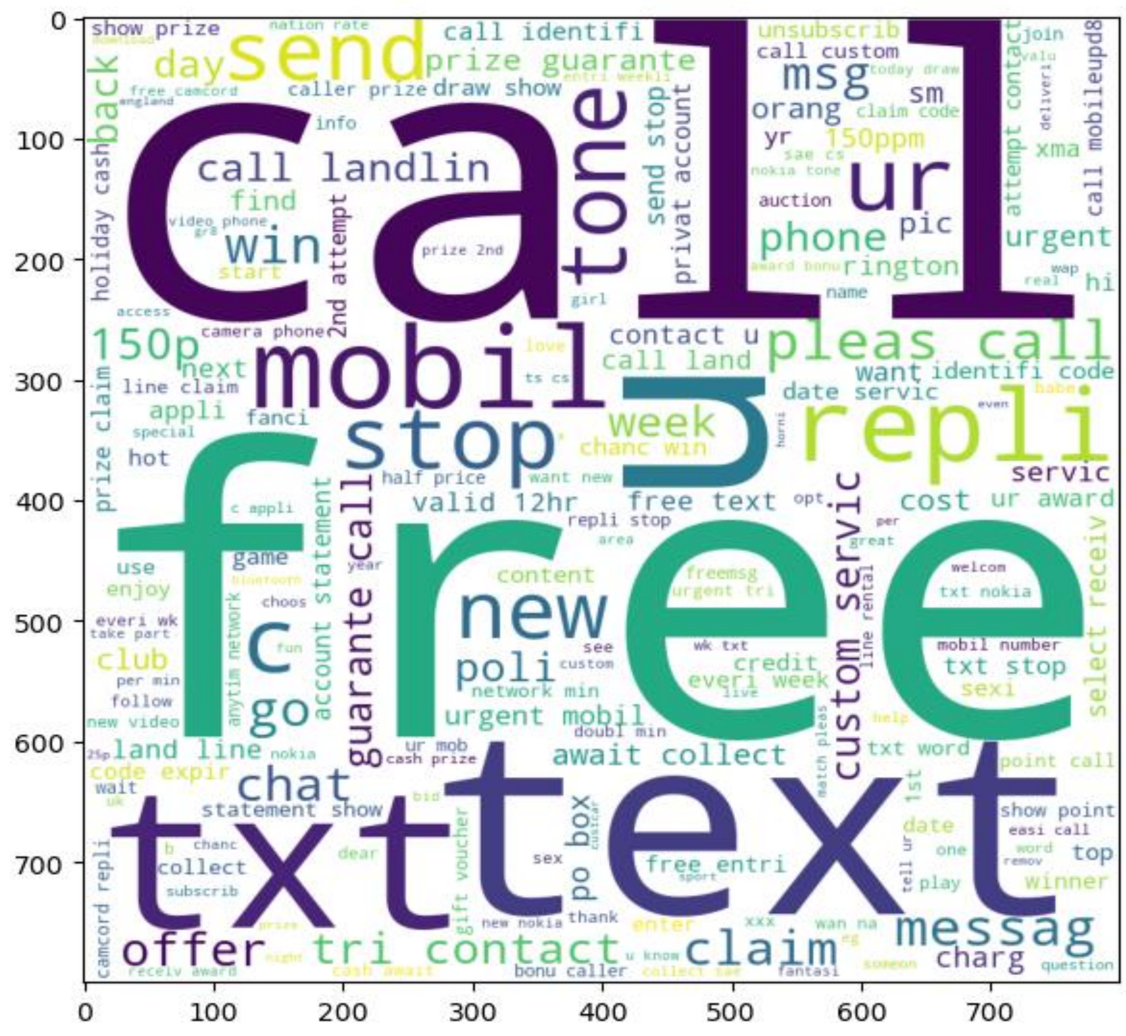
Word Cloud For Spam Messeges

```
wc = WordCloud(width=800, height=800, min_font_size=10, background_color='white')
```

```
spam_wc = wc.generate(df_spam[df_spam['target']==1]['transformed_text'].str.cat(sep=" "))
```

```
plt.figure(figsize=(10,7))
plt.imshow(spam_wc)
```

```
<matplotlib.image.AxesImage at 0x1adf410b250>
```



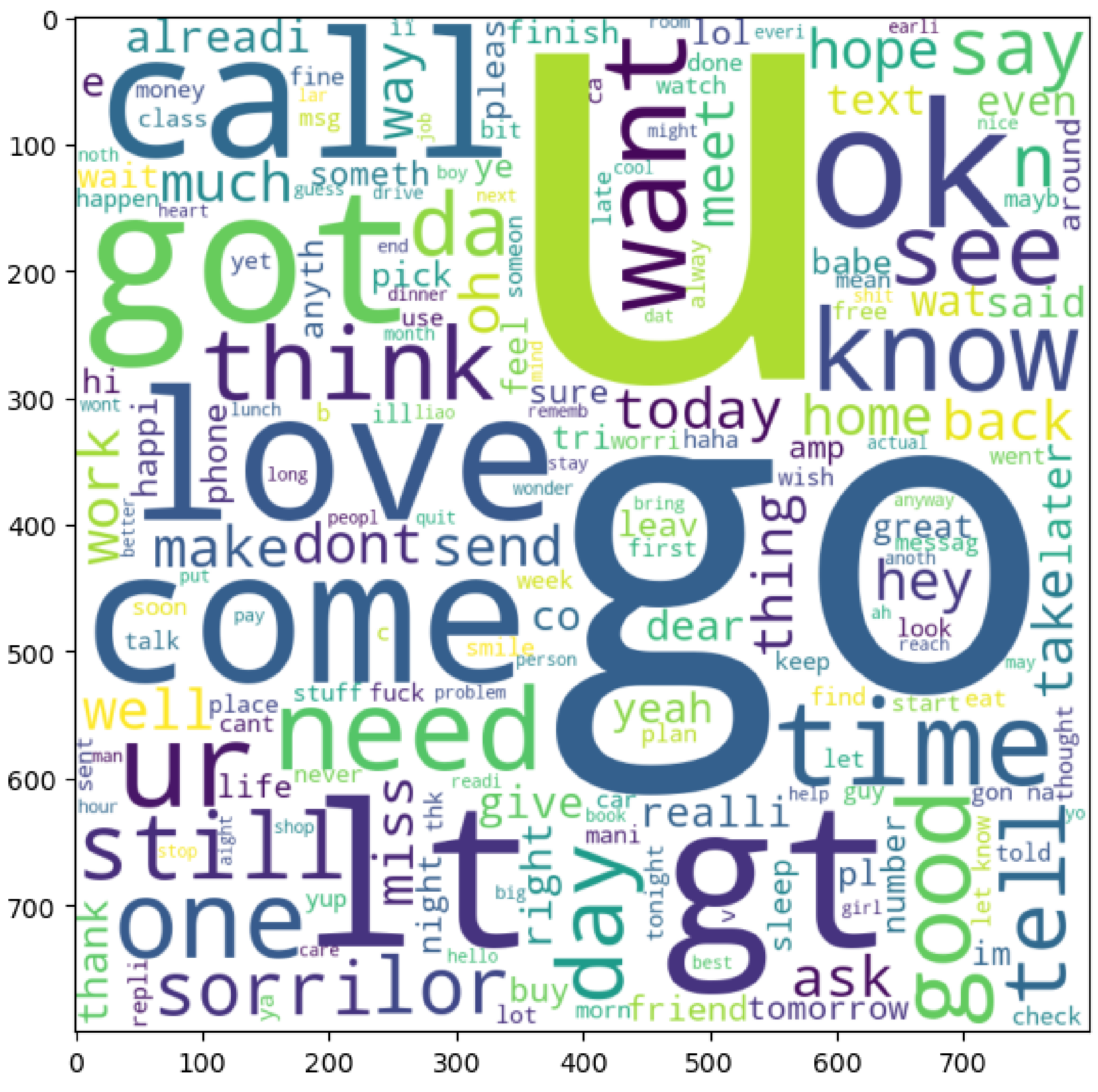
In this word cloud image, we can see the most word used in spam messages are called, free, text, claim, stop, mobile etc.

Word Cloud For Ham Messeges

```
ham_wc = wc.generate(df_spam[df_spam['target']==0]['transformed_text'].str.cat(sep=" "))
```

```
plt.figure(figsize=(10,7))
plt.imshow(ham_wc)
```

```
<matplotlib.image.AxesImage at 0x1adf5a29160>
```



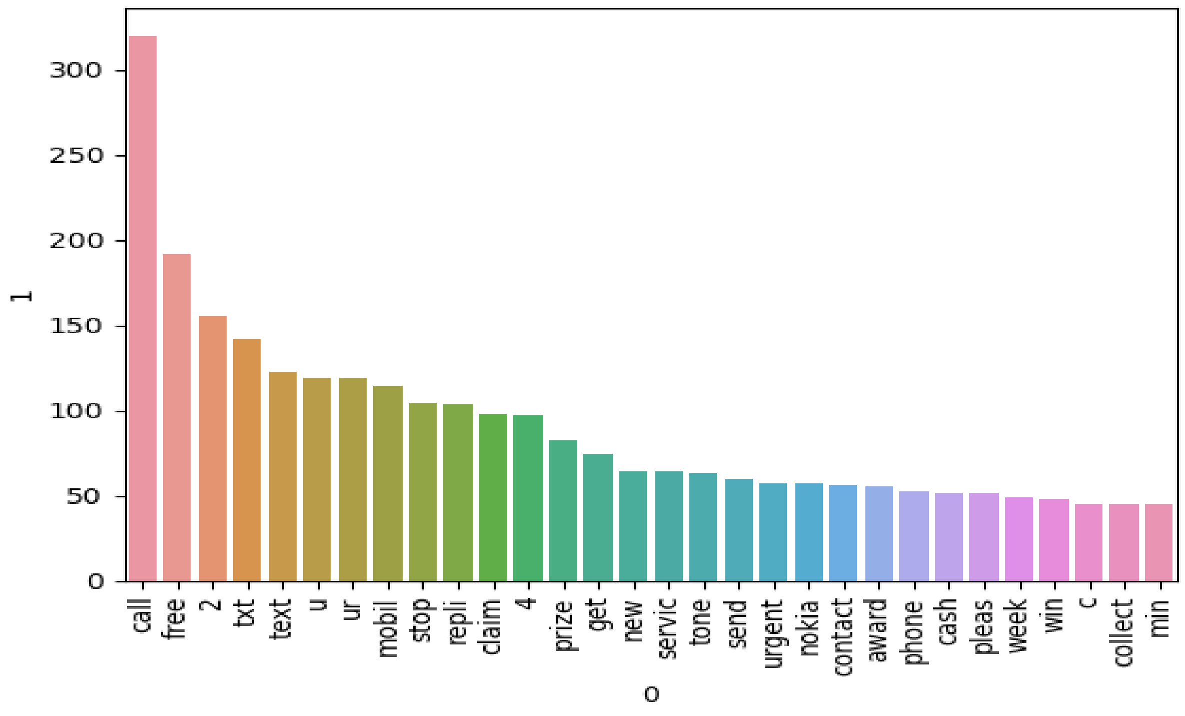
In this word cloud image, we can see the mostly word used in ham messages are love, come, go, need, time, good, call, u, want etc.

Import Counter

A Counter is a dict subclass for counting hashable objects. It is a collection where elements are stored as dictionary keys and their counts are stored as dictionary values. Counts are allowed to be any integer value including zero or negative counts.

```
from collections import Counter
```

```
sns.barplot(pd.DataFrame(Counter(spam_corpus).most_common(30))[0], pd.DataFrame(Counter(spam_corpus).most_common(30))[1])
plt.xticks(rotation='vertical')
plt.show()
```



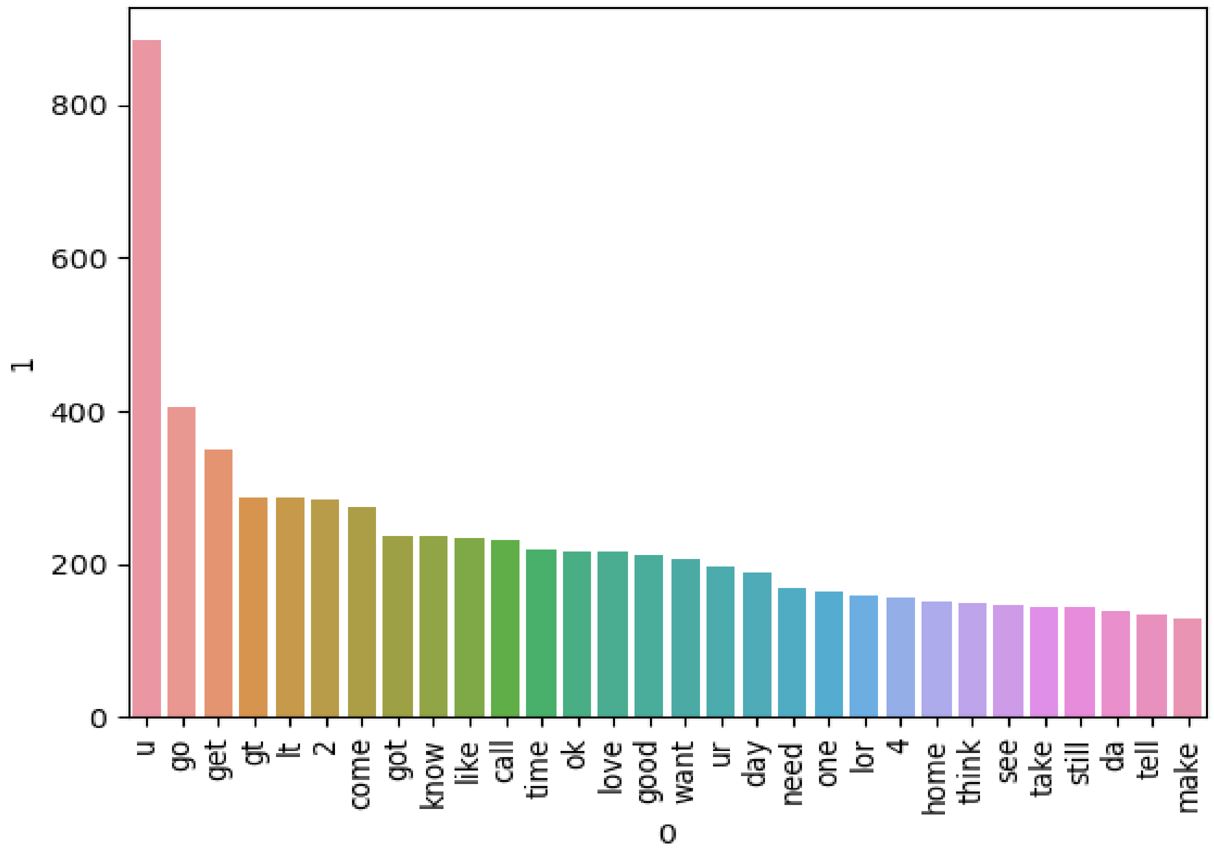
In this bar plot we can see most common 30 words use in spam messages.

```
# Create empty list ham_corpus for collecting the words in ham messeges.

ham_corpus=[]
for msg1 in df_spam[df_spam['target']==0]['transformed_text'].tolist():
    for word1 in msg1.split():
        ham_corpus.append(word1)
```

```
len(ham_corpus)
```

```
35394
```

```
sns.barplot(pd.DataFrame(Counter(ham_corpus).most_common(30))[0], pd.DataFrame(Counter(ham_corpus).most_common(30))[1])
plt.xticks(rotation='vertical')
plt.show()
```



In this bar plot, we can see the most common 30 words used in ham messages.

Project Report on

EMAIL SPAM DETECTION

Hardware & Software Requirements Tool Used

Hardware Used:

Processor — AMD Ryzen 5
RAM - 8 GB
ROM – 512 GB SSD
4GB Nvidia GEFORCE GTX Graphics card

Software utilized:

Anaconda – Jupyter Notebook

Models Development & Evaluation

IDENTIFICATION OF POSSIBLE PROBLEM-SOLVING APPROACHES:

- Our objective is to detect **spam messages** and analyse whether the message is ham or spam. This problem can be solved using Classification-based machine learning algorithms like Naïve Bayes Classifiers. For that purpose, the first task is to convert text data into numerical features with the help of the Vectorization Method.
- The final model is built over this scaled data. For building the ML model before implementing the classification algorithm, data is split into training & test data using train_test_split from the model selection module of the sklearn library.

Model Building

Convert Text Data into Numeric

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
cv=CountVectorizer()
tfidf=TfidfVectorizer(max_features=3000)
```

```
X = tfidf.fit_transform(df_spam['transformed_text']).toarray()
X
```

```
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.]])
```

Convert text messeges in to numeric form with the help of TfidfVectorizer(max_features=3000)

The purpose of max_features is to limit the number of features (words) from the dataset for which we want to calculate the TF-IDF scores. This is done by choosing the features based on term frequency across the corpus.

The TfidfVectorizer converts a collection of raw documents into a matrix of TF-IDF features.

```
X.shape
```

```
(5169, 3000)
```

```
y = df_spam['target'].values
y
```

```
array([0, 0, 1, ..., 0, 0, 0])
```

```
y.shape
```

```
(5169,)
```

Splitting The Data For Traning and Testing

```
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, random_state=2)
```

Testing of Identified Approaches (Algorithms)

The different classification algorithms used in this project to build the ML model are as below:

- Naïve Bayes Classifier
  - GaussianNB
  - MultinomialNB
  - BernoulliNB
- Random Forest Classifier
- Decision Tree Classifier
- SVC (Support Vector Classifier)

RUN AND EVALUATE SELECTED MODELS

Naïve Bayes Classifier:

```
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
```

```
gnb = GaussianNB()
mnb = MultinomialNB()
bnb = BernoulliNB()
```

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

a) GaussianNB

Model Fitting

```
gnb.fit(X_train, y_train)
y_pred1 = gnb.predict(X_test)
```

Accuracy Score

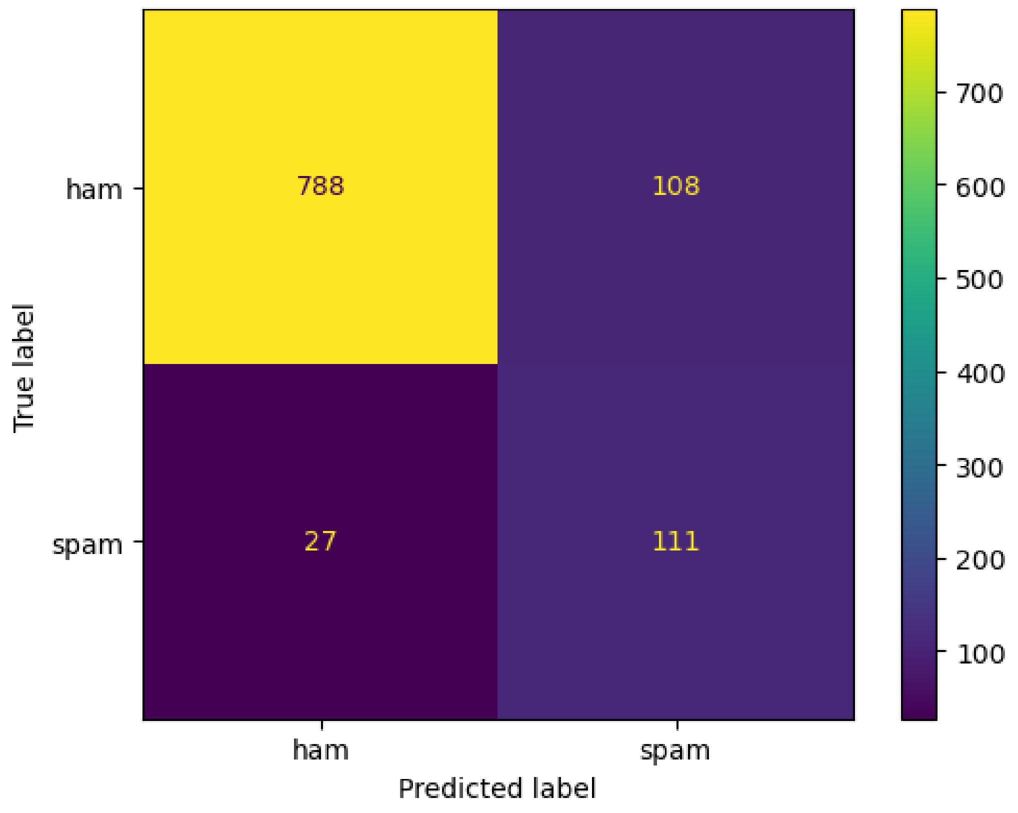
```
print('\33[1m' + "GaussianNB accuracy_score : {}%".format(round(accuracy_score(y_test, y_pred1)*100, 2)))
```

```
GaussianNB accuracy_score : 86.94%
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred1)
cmd = ConfusionMatrixDisplay(cm, display_labels=['ham','spam'])
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1adf5f26a60>
```



Precision Score

```
print('\33[1m' + "GaussianNB precision_score : {}%".format(round(precision_score(y_test, y_pred1)*100, 2)))
```

```
GaussianNB precision_score : 50.68%
```

b) MultinomialNB

Model Fitting

```
mnb.fit(X_train, y_train)
y_pred2 = mnb.predict(X_test)
```

Accuracy Score

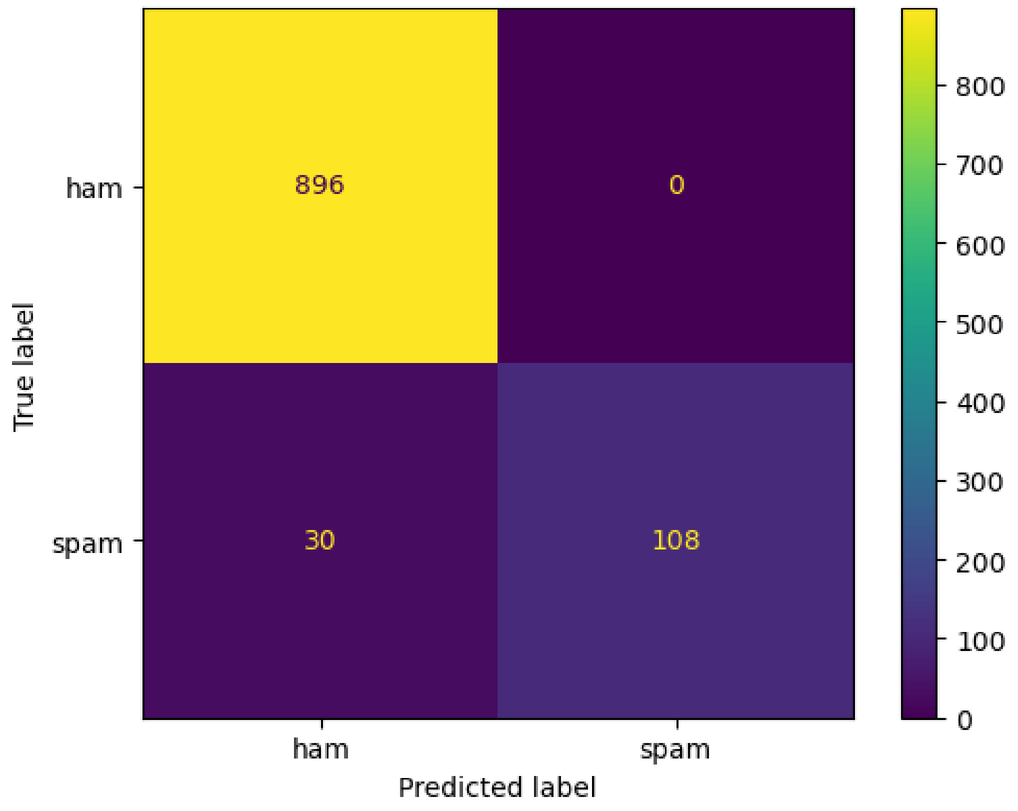
```
print('\33[1m' + "MultinomialNB accuracy_score : {}%".format(round(accuracy_score(y_test, y_pred2)*100, 2)))
```

```
MultinomialNB accuracy_score : 97.1%
```

Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred2)
cmd = ConfusionMatrixDisplay(cm, display_labels=['ham','spam'])
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1adf47751f0>
```



Precision Score

```
print('\33[1m' + "MultinomialNB precision_score : {}%".format(round(precision_score(y_test, y_pred2)*100, 2)))
```

```
MultinomialNB precision_score : 100.0%
```

c) BernoulliNB

Model Fitting

```
bnb.fit(X_train, y_train)
y_pred3 = bnb.predict(X_test)
```

Accuracy Score

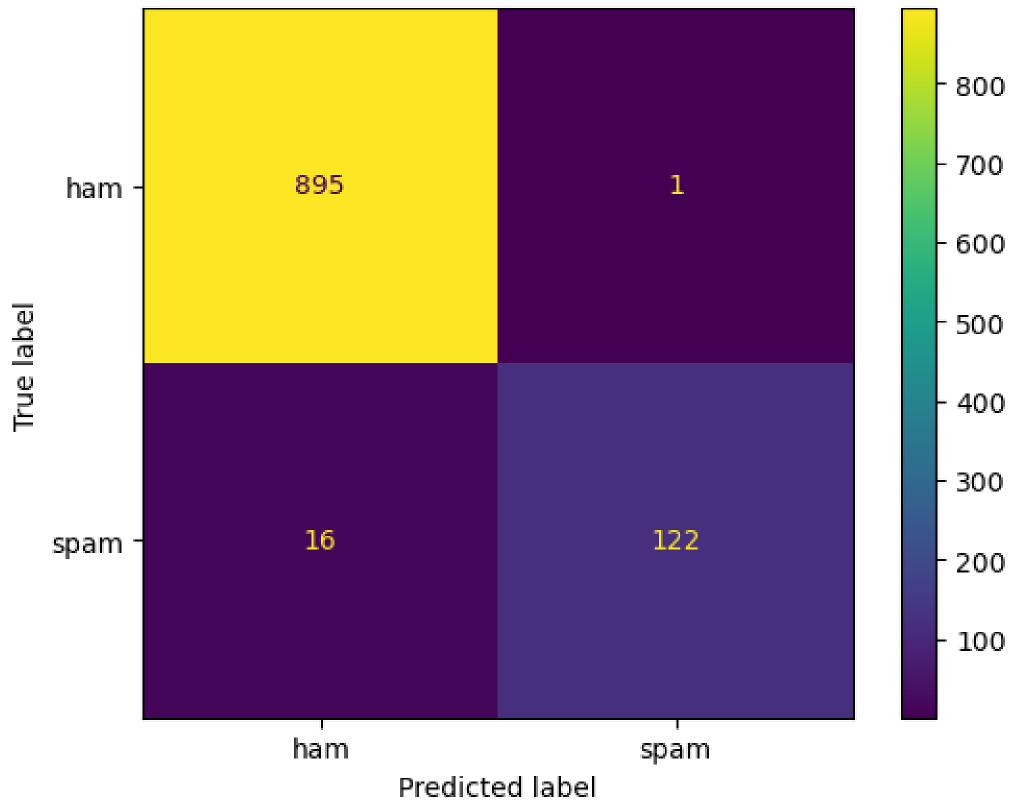
```
print('\33[1m' + "BernoulliNB accuracy_score : {}%".format(round(accuracy_score(y_test, y_pred3)*100, 2)))
```

```
BernoulliNB accuracy_score : 98.36%
```

Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred3)
cmd = ConfusionMatrixDisplay(cm, display_labels=['ham','spam'])
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1adf5f01fa0>
```



Precision Score

```
print('\33[1m' + "BernoulliNB precision_score : {}%".format(round(precision_score(y_test, y_pred3)*100, 2)))
```

```
BernoulliNB precision_score : 99.19%
```

Decision Tree Classifier

Model Fitting

```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
dtc_pred = dtc.predict(X_test)
```

Accuracy Score(DecisionTreeClassifier)

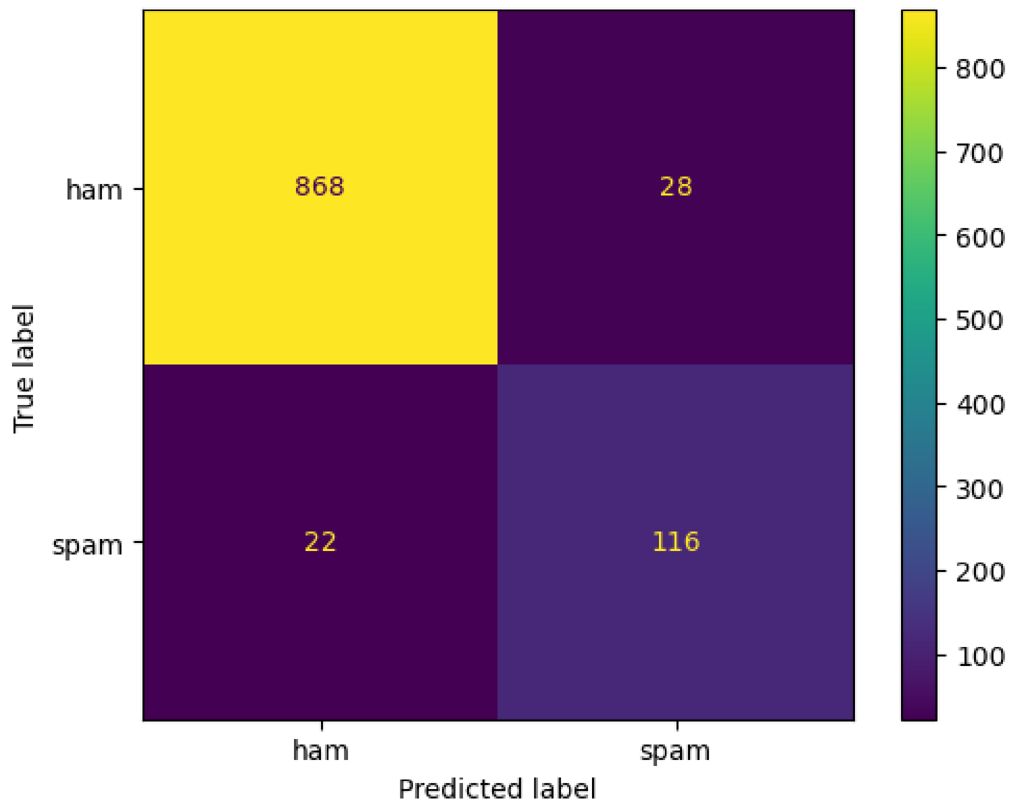
```
print('\33[1m' + "Decision Tree accuracy_score : {}%".format(round(accuracy_score(y_test, dtc_pred)*100, 2)))
```

```
Decision Tree accuracy_score : 95.16%
```

Confusion Matrix(DecisionTreeClassifier)

```
cm = confusion_matrix(y_test, dtc_pred)
cmd = ConfusionMatrixDisplay(cm, display_labels=['ham','spam'])
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1ad85ffc7f0>
```



Precision Score(DecisionTreeClassifier)

```
print('\33[1m' + "Decision Tree precision_score : {}%".format(round(precision_score(y_test, dtc_pred)*100, 2)))
```

```
Decision Tree precision_score : 80.56%
```

Random Forest Classifier

Model Fitting(RandomForestClassifier)

```
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
rfc_pred = rfc.predict(X_test)
```

Accuracy Score(RandomForestClassifier)

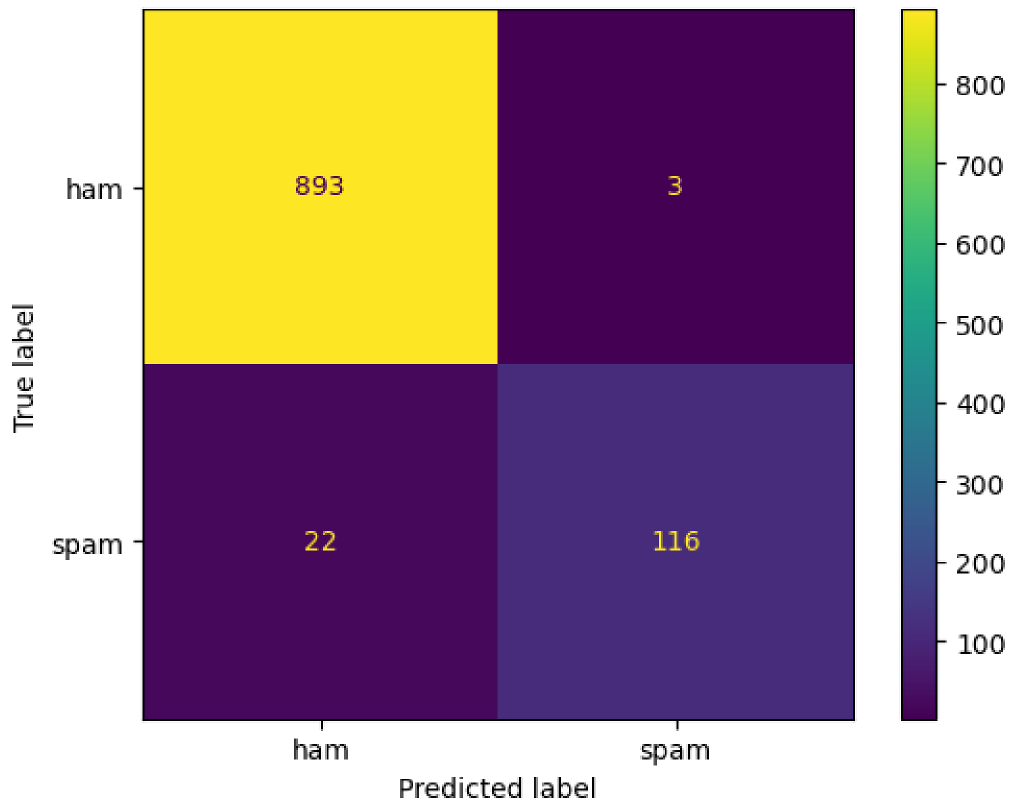
```
print('\33[1m' + "Random Forest accuracy_score : {}%".format(round(accuracy_score(y_test, rfc_pred)*100, 2)))
```

```
Random Forest accuracy_score : 97.58%
```

Confusion Matrix(RandomForestClassifier)

```
cm = confusion_matrix(y_test, rfc_pred)
cmd = ConfusionMatrixDisplay(cm, display_labels=['ham','spam'])
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1adf5ac6be0>
```



Precision Score(RandomForestClassifier)

```
print('\33[1m' + "Random Forest precision_score : {}%".format(round(precision_score(y_test, rfc_pred)*100, 2)))
```

```
Random Forest precision_score : 97.48%
```

SVC

Model Fitting(SVC)

```
svc = SVC()
svc.fit(X_train, y_train)
svc_pred = svc.predict(X_test)
```

Accuracy Score(SVC)

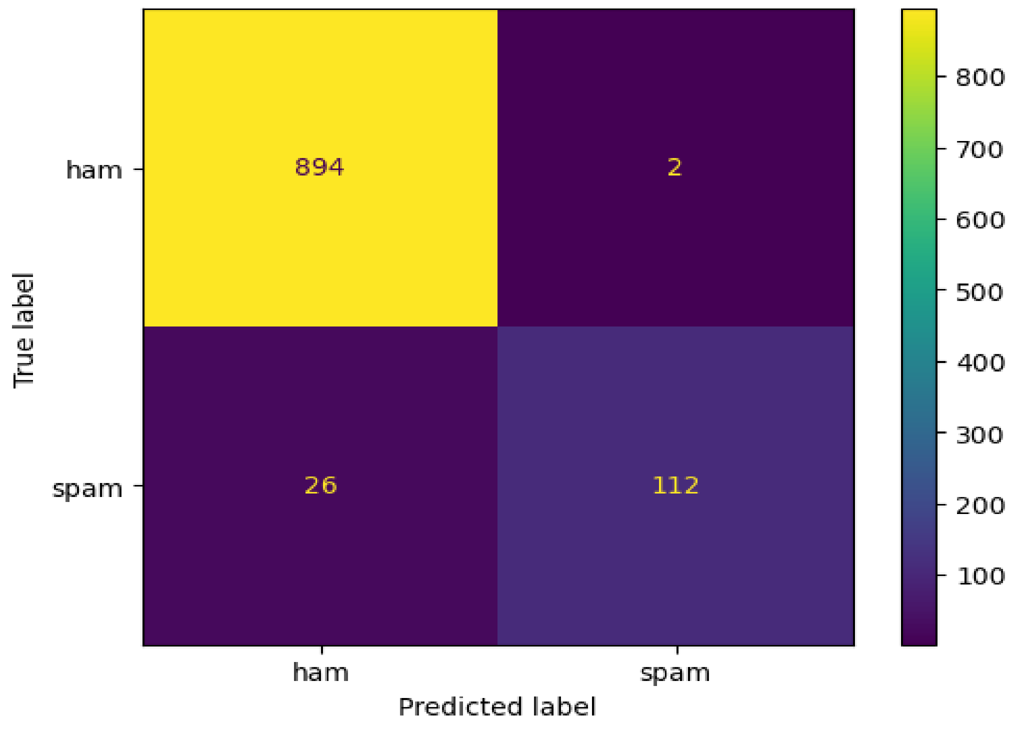
```
print('\33[1m' + "SVC accuracy_score : {}%".format(round(accuracy_score(y_test, svc_pred)*100, 2)))
```

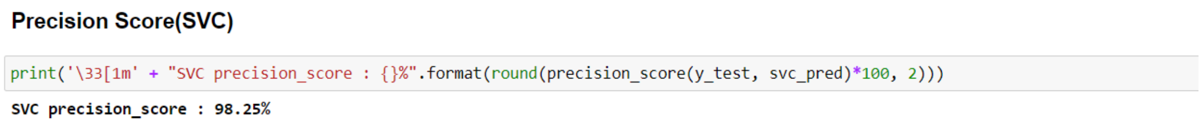
```
SVC accuracy_score : 97.29%
```

Confusion Matrix(SVC)

```
cm = confusion_matrix(y_test, svc_pred)
cmd = ConfusionMatrixDisplay(cm, display_labels=['ham','spam'])
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1ad86122d00>
```





Precision Score(SVC)

```
print('\33[1m' + "SVC precision_score : {}%".format(round(precision_score(y_test, svc_pred)*100, 2)))
```

```
SVC precision_score : 98.25%
```

Algorithms Use in This Model:

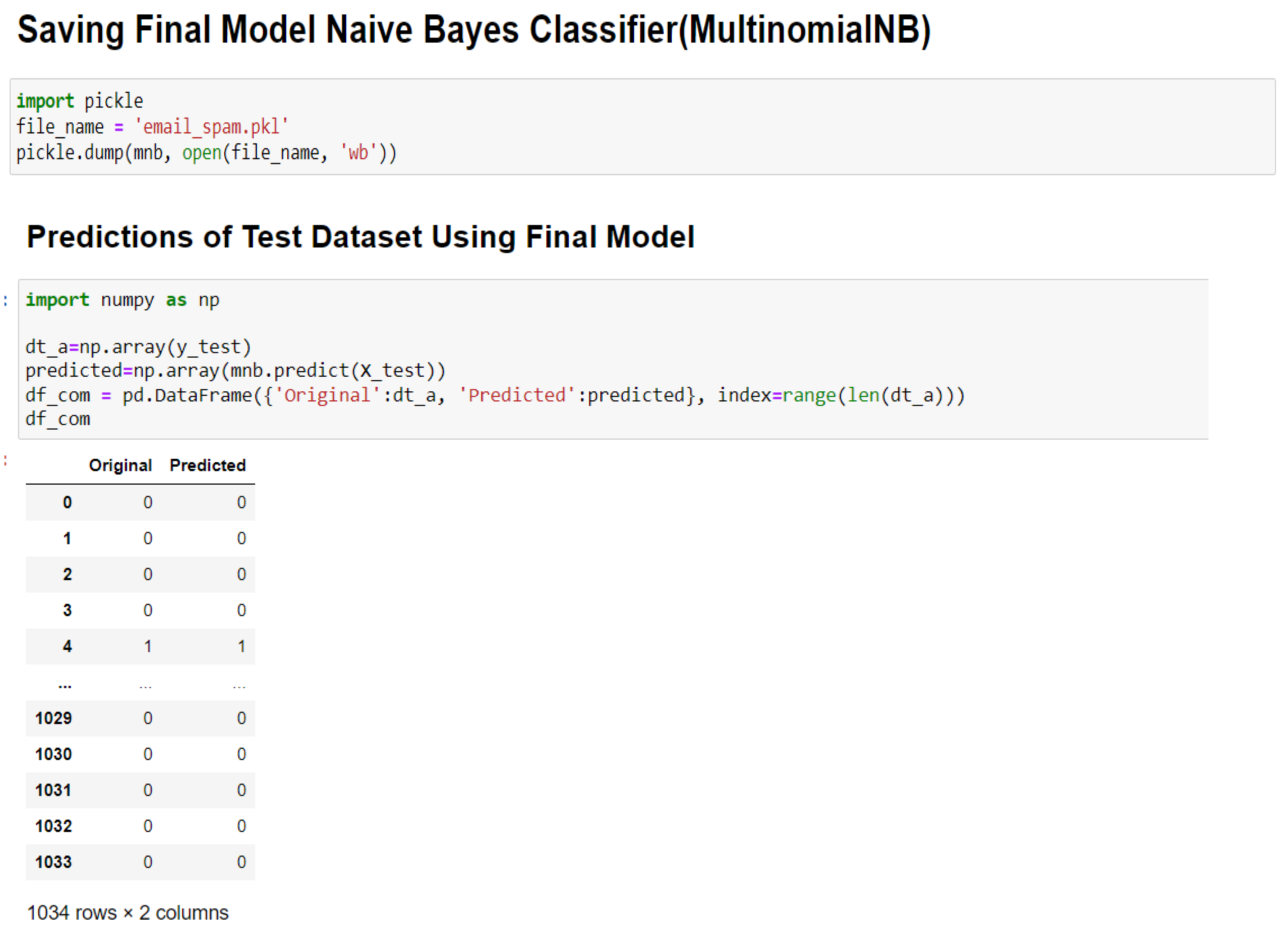
ALGORITHMS NAME	ACCURACY SCORE	PRECISION SCORE
1. Naive Bayes Classifier		
a) Gaussian NB	86.94%	50.68%
b) Multinomial NB	97.1%	100.0%
c) Bernoulli NB	98.36%	99.19%
2. Decision Tree Classifier	95.16%	80.56%
3. RandomForest Classifier	97.58%	97.48%
4. Support Vector Classifier (SVC)	97.29%	98.25%

- **We can see that in Naive Bayes Classifier the MultinomialNB and BernoulliNB both are best accuracy score and precision score but MultinomialNB is higher precision score 100%**

- **The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.**

EMAIL SPAM DETECTION

- The best value is 1 and the worst value is 0.
- MultinomialNB is a Final Model because of the precision score is 100.0%



Saving Final Model Naive Bayes Classifier(MultinomialNB)

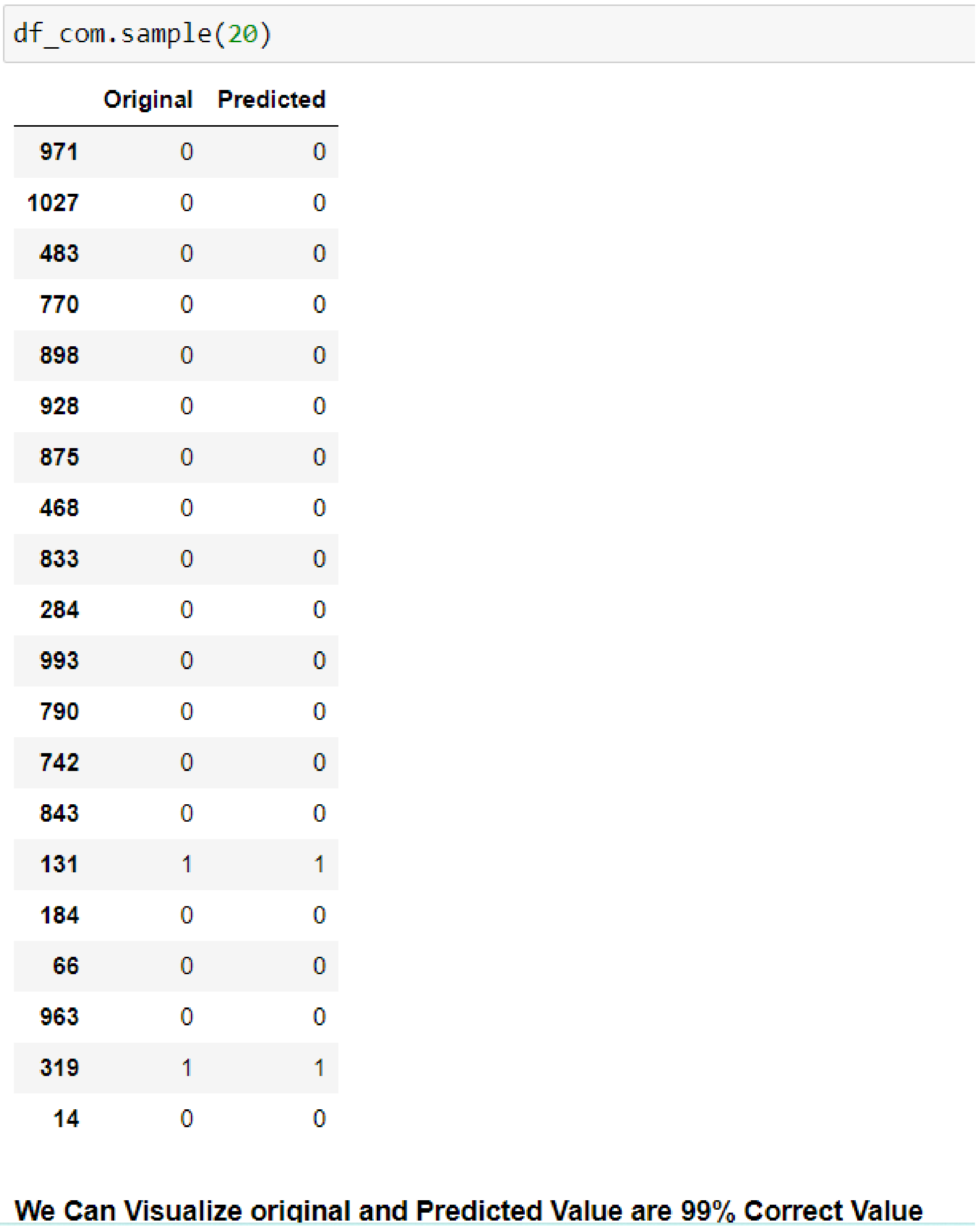
```python
import pickle
file_name = 'email_spam.pkl'
pickle.dump(mnb, open(file_name, 'wb'))
```

Predictions of Test Dataset Using Final Model

```python
import numpy as np

dt_a=np.array(y_test)
predicted=np.array(mnb.predict(X_test))
df_com = pd.DataFrame({'Original':dt_a, 'Predicted':predicted}, index=range(len(dt_a)))
df_com
```

	Original	Predicted
0	0	0
1	0	0
2	0	0
3	0	0
4	1	1
...	...	...
1029	0	0
1030	0	0
1031	0	0
1032	0	0
1033	0	0

1034 rows × 2 columns



```
df_com.sample(20)
```

	Original	Predicted
971	0	0
1027	0	0
483	0	0
770	0	0
898	0	0
928	0	0
875	0	0
468	0	0
833	0	0
284	0	0
993	0	0
790	0	0
742	0	0
843	0	0
131	1	1
184	0	0
66	0	0
963	0	0
319	1	1
14	0	0

We Can Visualize original and Predicted Value are 99% Correct Value

Project Report on

EMAIL SPAM DETECTION

Conclusion

Key Findings and Conclusions of the Study

ALGORITHMS NAME	ACCURACY SCORE	PRECISION SCORE
1. Naive Bayes Classifier		
a) Gaussian NB	86.94%	50.68%
b) Multinomial NB	97.1%	100.0%
c) Bernoulli NB	98.36%	99.19%
2. Decision Tree Classifier	95.16%	80.56%
3. RandomForest Classifier	97.58%	97.48%
4. Support Vector Classifier (SVC)	97.29%	98.25%

We can see that in Naive Bayes Classifier the MultinomialNB and BernoulliNB both are best accuracy score and precision score but MultinomialNB is higher precision score 100%

The best value is 1 and the worst value is 0.

MultinomialNB is a Final Model because of the precision score is 100.0%