Steps of project:-

1. Loading Data
2. Data cleaning
3. EDA
4. Text Preprocessing
5. Model Building
6. Evaluation
7. Improvement Depending on the evaluation
8. Convert into Website
9. Deploy

In [117…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

1. Loading Data

In [118…
```
df = pd.read_csv('spam.csv',encoding='latin1')
```

In [119…
```
# df.head()
df.sample(5)
```

Out[119]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
3017	ham	I didn't get the second half of that message	NaN	NaN	NaN
1900	ham	And miss vday the parachute and double coins??...	NaN	NaN	NaN
2160	ham	No. Its not specialisation. Can work but its s...	NaN	NaN	NaN
4988	ham	So your telling me I coulda been your real Val...	NaN	NaN	NaN
2421	ham	Err... Cud do. I'm going to at 8pm. I haven't...	NaN	NaN	NaN

In [120…
```
df.shape
```

Out[120]:
```
(5572, 5)
```

In [ ]:
```
```

2. Data Cleaning

In [121…
```
df.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
```

In [122…
```
# drop last 3 cols
df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace=True)
```

In [123…
```
df.sample(5)
```

Out[123]:

	v1	v2
5024	ham	I was gonna ask you lol but i think its at 7
2961	ham	\NONE!NOWHERE IKNO DOESDISCOUNT!SHITINNIT\""
411	ham	Cos i want it to be your thing
2911	ham	You didn't have to tell me that...now i'm thin...
4424	ham	Just now saw your message.it k da:)

In [124…
```
# renaming the cols
df.rename(columns={'v1':'target','v2':'text'},inplace=True)
df.sample(5)
```

Out[124]:

	target	text
4558	ham	Think + da. You wil do.
3612	ham	Depends on individual lor e hair dresser say p...
4410	ham	Ya but it cant display internal subs so i gott...
1907	ham	ELLO BABE U OK?
754	ham	Realy sorry-i don't recognise this number and ...

In [125…
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df['target'] = encoder.fit_transform(df['target'])
```

In [126…
```
df.head()
```

Out[126]:

	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...

In [127…
```
# missing values
df.isnull().sum()
```

Out[127]:
```
target    0
text      0
dtype: int64
```

In [128…
```
# check for duplicate values
print(df.duplicated().sum())
```
```
403
```

In [129…
```
# remove duplicates
df = df.drop_duplicates(keep='first')
```

In [130…
```
# Then we again check for duplicate values
print(df.duplicated().sum())
```
```
0
```

In [131…
```
df.shape
```

Out[131]:
```
(5169, 2)
```

In [ ]:
```
```

3. (EDA exploratory data analysis)

In [132…
```
df.head()
```

Out[132]:

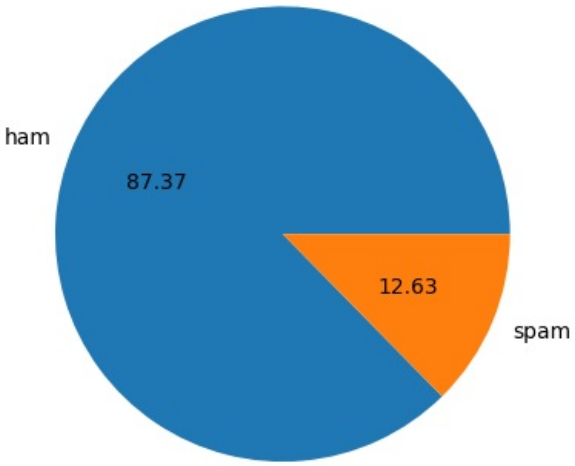
	target	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

In [133…
```
df['target'].value_counts()
```

In [133]:
```
df['target'].value_counts()
```

Out[133]:
```
target
0    4516
1     653
Name: count, dtype: int64
```

In [134]:
```
import matplotlib.pyplot as plt
plt.pie(df['target'].value_counts(), labels=['ham','spam'],autopct="%0.2f")
plt.show()
```



In [135]:
```
# Data is imbalanced
```

In [136]:
```
import nltk
```

In [137]:
```
nltk.download('punkt')
```

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

Out[137]:
```
True
```

Now we need to do deeper analysis then we create three new columns

1. Number of character in the sms
2. Number of word in the sms
3. Number of sentences in the sms

In [138]:
```
df['num_char'] = df['text'].apply(len)
```

In [139]:
```
df.head()
```

Out[139]:

	target	text	num_char
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

Word_tokenize function in nltk==>>>This function breaks a sentence into individual words or tokens.

In [140]:
```
# num of words
df['num_words'] = df['text'].apply(lambda x:len(nltk.word_tokenize(x)))
df['num_words']
```

Out[140]:
```
0       24
1        8
2       37
3       13
4       15
        ..
5567    35
5568     9
5569    15
5570    27
5571     7
Name: num_words, Length: 5169, dtype: int64
```

In [141…
```
df.head()
```

Out[141]:

	target	text	num_char	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

Sent_tokenize function in nltk===>>> This function splits a piece of text into individual sentences.

In [142…
```
df['num_sentenc'] = df['text'].apply(lambda x:len(nltk.sent_tokenize(x)))
```

In [143…
```
df.head()
```

Out[143]:

	target	text	num_char	num_words	num_sentenc
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

In [144…
```
df[['num_char','num_words','num_sentenc']].describe()
```

Out[144]:

	num_char	num_words	num_sentenc
count	5169.000000	5169.000000	5169.000000
mean	78.977945	18.455794	1.965564
std	58.236293	13.324758	1.448541
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	38.000000

Now we can also analyse ham and sam in different way then we write the code

In [145…
```
#Ham messages
df[df['target']==0][['num_char','num_words','num_sentenc']].describe()
```

Out[145]:

	num_char	num_words	num_sentenc
count	4516.000000	4516.000000	4516.000000
mean	70.459256	17.123782	1.820195
std	56.358207	13.493970	1.383657
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	38.000000

In [146…
```
#Spam messages
```

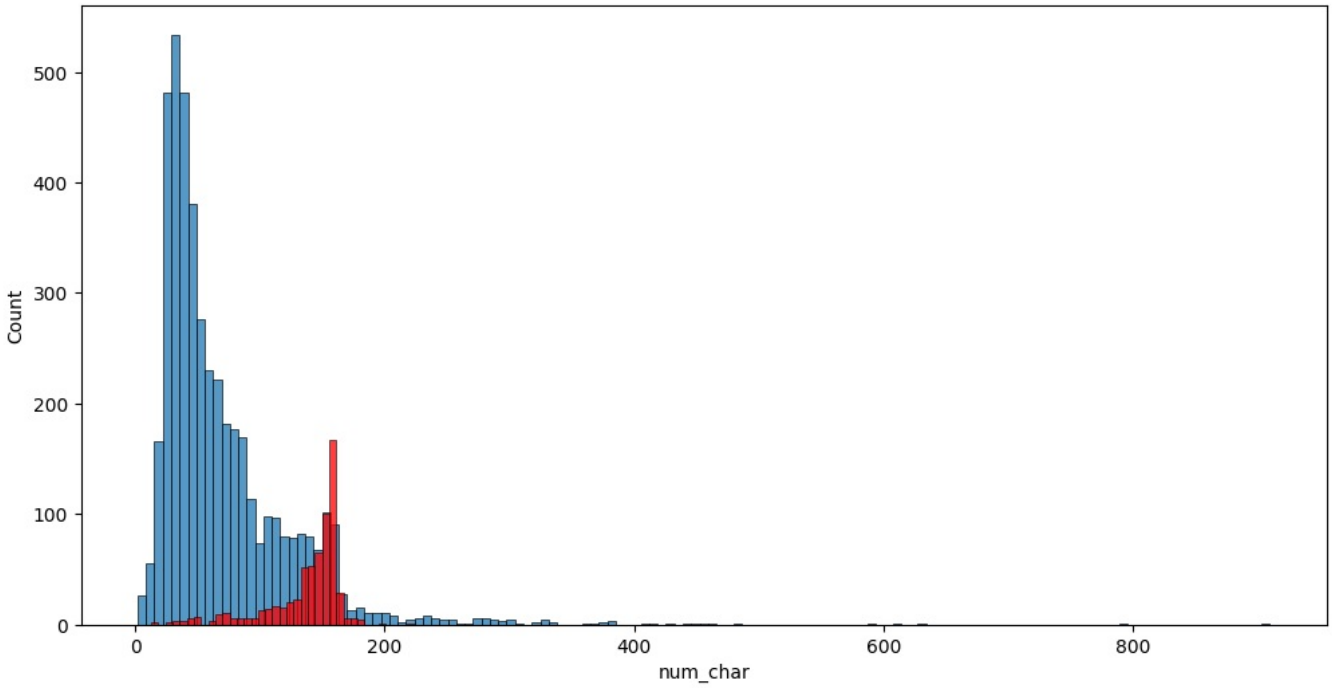
```
df[df['target']==1][['num_char','num_words','num_sentenc']].describe()
```

Out[146]:

	num_char	num_words	num_sentenc
count	653.000000	653.000000	653.000000
mean	137.891271	27.667688	2.970904
std	30.137753	7.008418	1.488425
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	9.000000

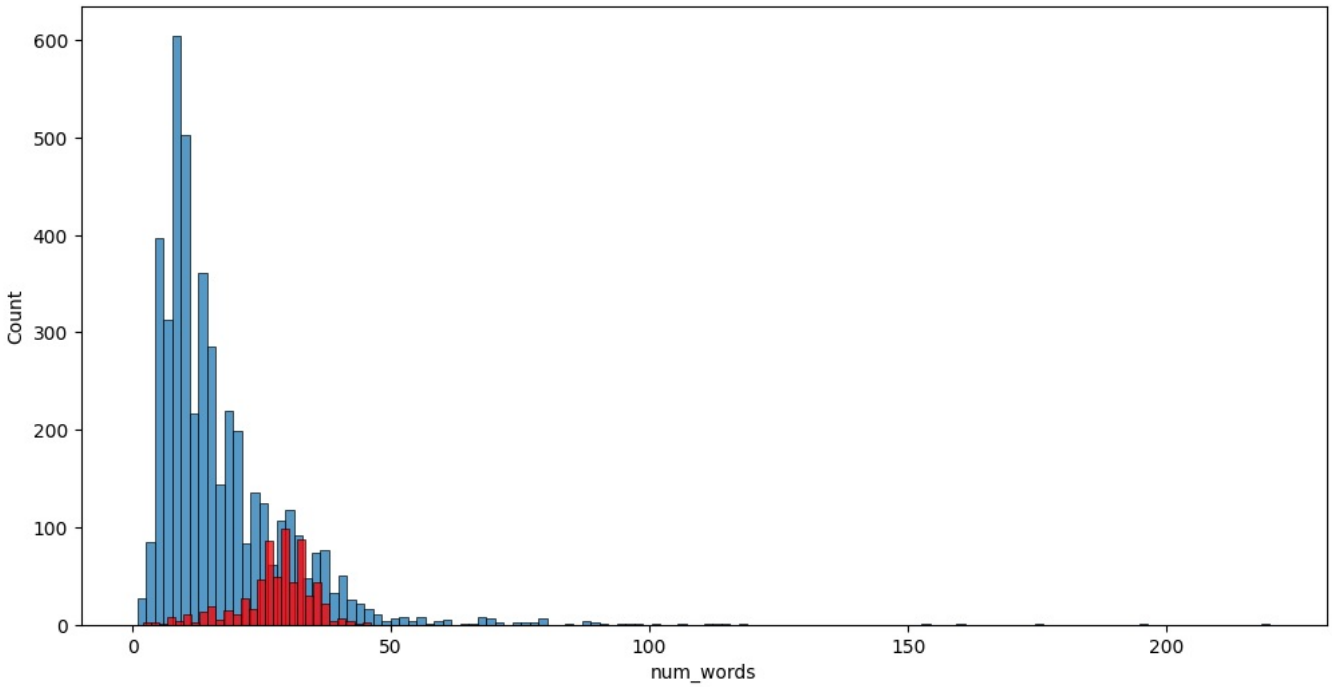
In [147…
```
#Visualization Ham and spam messages
plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_char'])
sns.histplot(df[df['target'] == 1]['num_char'],color='red')
```

Out[147]:
```
<Axes: xlabel='num_char', ylabel='Count'>
```



In [148…
```
#Visualization Ham and spam messages in the terms are the number of word
plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_words'])
sns.histplot(df[df['target'] == 1]['num_words'],color='red')
```

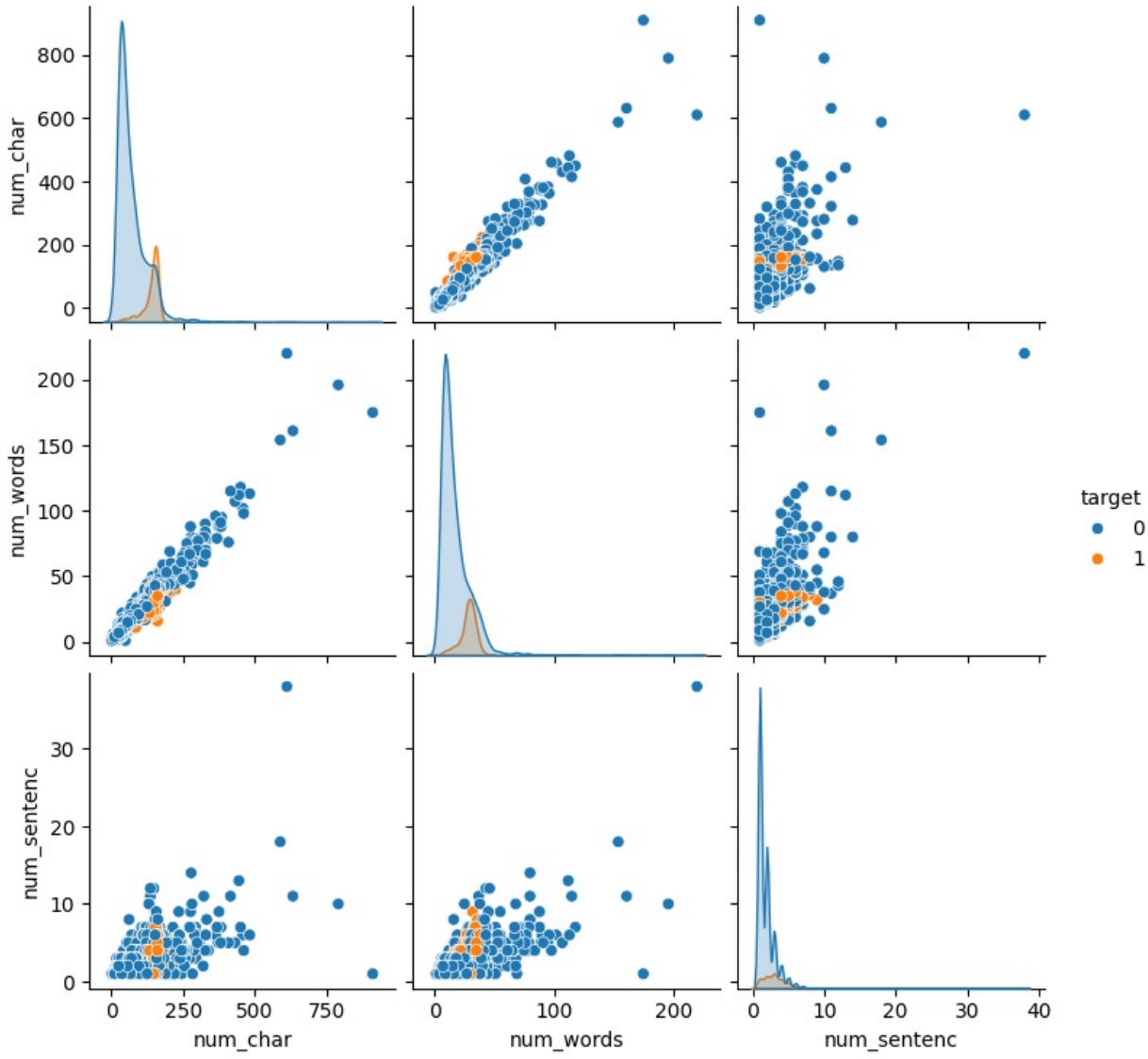
Out[148]:
```
<Axes: xlabel='num_words', ylabel='Count'>
```



We can clearly analyse that the more ham message is created by using less no of word and more spam message is created by using more number of word

In [149…
```
# "Let's see how these columns are related to each other, and then we will visualize them using a pairplot.
sns.pairplot(df,hue='target')
```

Out[149]:
```
<seaborn.axisgrid.PairGrid at 0x1e9a0aa45d0>
```


num_char
num_words
num_sentenc
target
0
1
0
250
500
750
0
100
200
0
10
20
30
40

In [150…

```
df
```

Out[150]:

	target	text	num_char	num_words	num_sentenc
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
...	...	...	...	...	...
5567	1	This is the 2nd time we have tried 2 contact u...	161	35	4
5568	0	Will Ì_ b going to esplanade fr home?	37	9	1
5569	0	Pity, * was in mood for that. So...any other s...	57	15	2
5570	0	The guy did some bitching but I acted like i'd...	125	27	1
5571	0	Rofl. Its true to its name	26	7	2

5169 rows × 5 columns

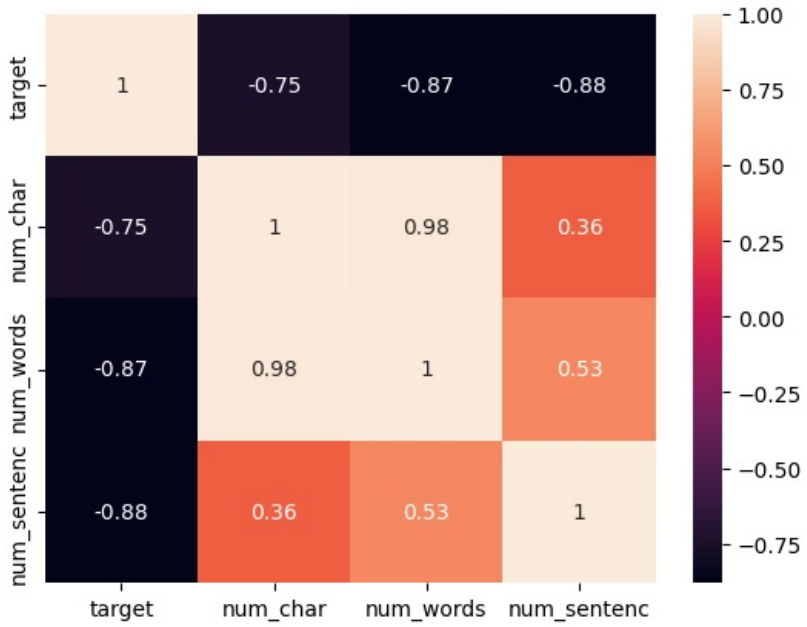
In [151…
```
#df.corr() is throw ValueError: could not convert string to float: 'Go until jurong point, crazy..Then we selec
# Select only numerical columns
numerical_df = df.select_dtypes(include=['float64', 'int64'])
```

In [152…
```
df_corrmatrix = numerical_df.corr()
print(df_corrmatrix)
```

```
              target  num_char  num_words  num_sentenc
target      1.000000  0.384717   0.262912     0.263939
num_char    0.384717  1.000000   0.965760     0.624139
num_words   0.262912  0.965760   1.000000     0.679971
num_sentenc 0.263939  0.624139   0.679971     1.000000
```

In [153…
```
sns.heatmap(df_corrmatrix.corr(),annot=True)
```

Out[153]:
```
<Axes: >
```



In [ ]:
```
```

4. Data Preprocessing or Text Preprocessing.

1. Lower case
2. Tokenization
3. Removing special characters
4. Removing stop words and punctuation
5. Stemming

In [154…
```
def transfrom_text(text):
    text=text.lower()#first case conver lower case
    text=nltk.word_tokenize(text)#second step tekenization
    return text
```

In [155…
```
transfrom_text('Hi How Are You')
```

```
transfrom_text('Hi How Are You')
```

Out[155]:
```
['hi', 'how', 'are', 'you']
```

In [156…
```
def transfrom_text(text):
    text=text.lower()#first case conver lower case
    text=nltk.word_tokenize(text)#second step tekenization
    y=[]
    for i in text:#Removing special characters
        if i.isalnum():
            y.append(i)
    return y
```

In [157…
```
transfrom_text('Hi Are You given your %%')
```

Out[157]:
```
['hi', 'are', 'you', 'given', 'your']
```

In [158…
```
from nltk.corpus import stopwords
# stopwords.words('english')#These way we check of stopword present in our datasets
```

In [159…
```
# # if we need to check punctuation then we write the code
import string
# string.punctuation
```

In [160…
```
from nltk.corpus import stopwords
import string
from nltk.stem import PorterStemmer
ps=PorterStemmer()
# ps.stem('dancing')#These way we implement the concept of steaming
```

In [161…
```
def transfrom_text(text):
    text=text.lower()#first case conver lower case
    text=nltk.word_tokenize(text)#second step tekenization
    y=[]
    for i in text:#Removing special characters
        if i.isalnum():
            y.append(i)
    text = y[:]
    y.clear()
    for i in text:#Removing stop words and punctuation
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)
    text = y[:]
    y.clear()

    for i in text:#Concept of stemming
        y.append(ps.stem(i))

    return " ".join(y)
```

In [162…
```
transfrom_text("I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried
```

Out[162]:
```
'gon na home soon want talk stuff anymor tonight k cri enough today'
```

In [163…
```
df['text'][10]
```

Out[163]:
```
"I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today.
"
```

In [164…
```
df['transfrom_text']=df['text'].apply(transfrom_text)
```

In [165…
```
df.head()
```

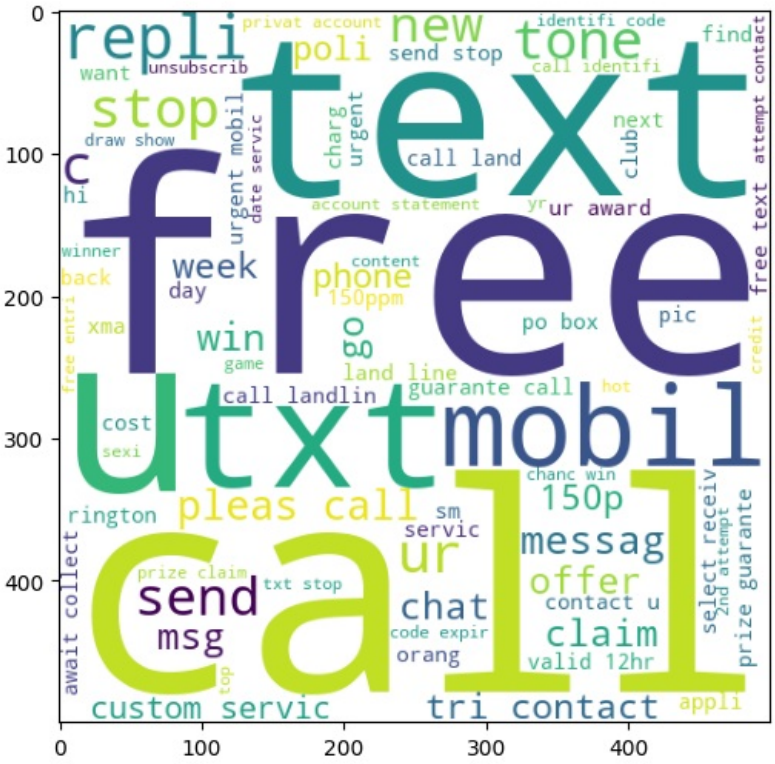
Out[165]:

	target	text	num_char	num_words	num_sentenc	transfrom_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

In [166…
```
# A WordCloud function in NLP (Natural Language Processing) is a tool that helps visualize the most frequent wo
from wordcloud import WordCloud#and in this case we visualize the most frequent word that comes under the spam
wc = WordCloud(width=500,height=500,min_font_size=10,background_color="white")
```

In [167…
```
spam_wc=wc.generate(df[df['target']==1]['transfrom_text'].str.cat(sep=" "))
plt.figure(figsize=(15,6))
plt.imshow(spam_wc)
```

Out[167]:
```
<matplotlib.image.AxesImage at 0x1e9a48e4d10>
```



In [168…
```
ham_wc=wc.generate(df[df['target']==0]['transfrom_text'].str.cat(sep=" "))
plt.figure(figsize=(15,6))#and in this case we visualize the most frequent word that comes under the ham messag
plt.imshow(ham_wc)#Not spam message
```

Out[168]:
```
<matplotlib.image.AxesImage at 0x1e9a4e4b710>
```

In [169…
```
#lets see top 30 most frequent word in spam and ham messages
df.head()
```

Out[169]:

	target	text	num_char	num_words	num_sentenc	transfrom_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

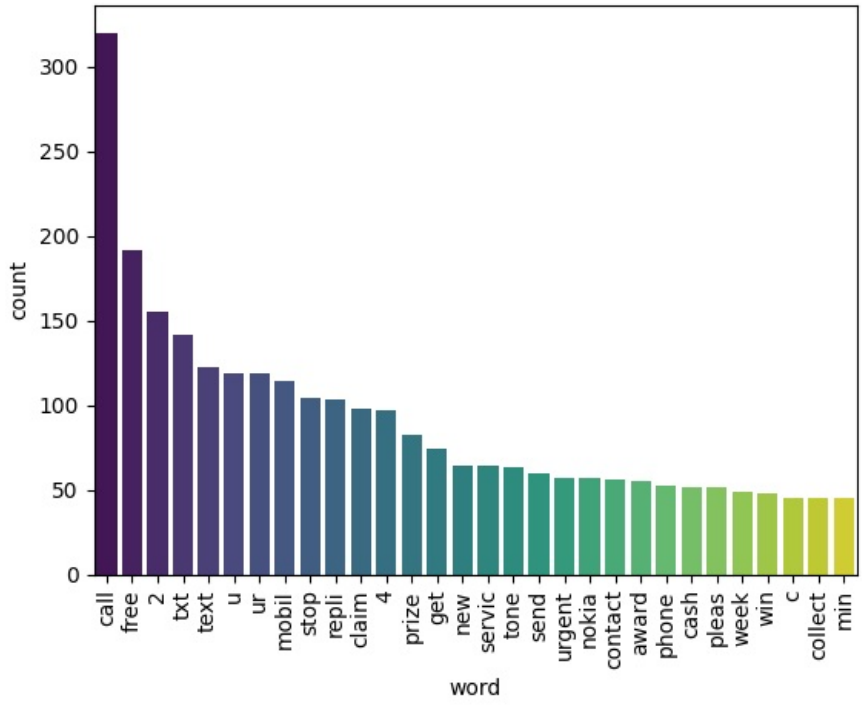
In [170…
```
#let's write the code and understand most frequently occurance 30 spam word or messages
```

```
spam_curpus=[]
for msg in df[df['target']==1]['transfrom_text'].tolist():
    for word in msg.split():
        spam_curpus.append(word)
```

In [171…
```
len(spam_curpus)
```

Out[171]:
```
9939
```

In [172…
```
from collections import Counter
# Create a DataFrame for the most common 30 words
word_counts = pd.DataFrame(Counter(spam_curpus).most_common(30), columns=['word', 'count'])
# Plot the barplot using x and y keyword arguments
sns.barplot(x='word', y='count', data=word_counts,palette='viridis')
# Rotate the x-axis labels for better readability
plt.xticks(rotation='vertical')
# Show the plot
plt.show()
```

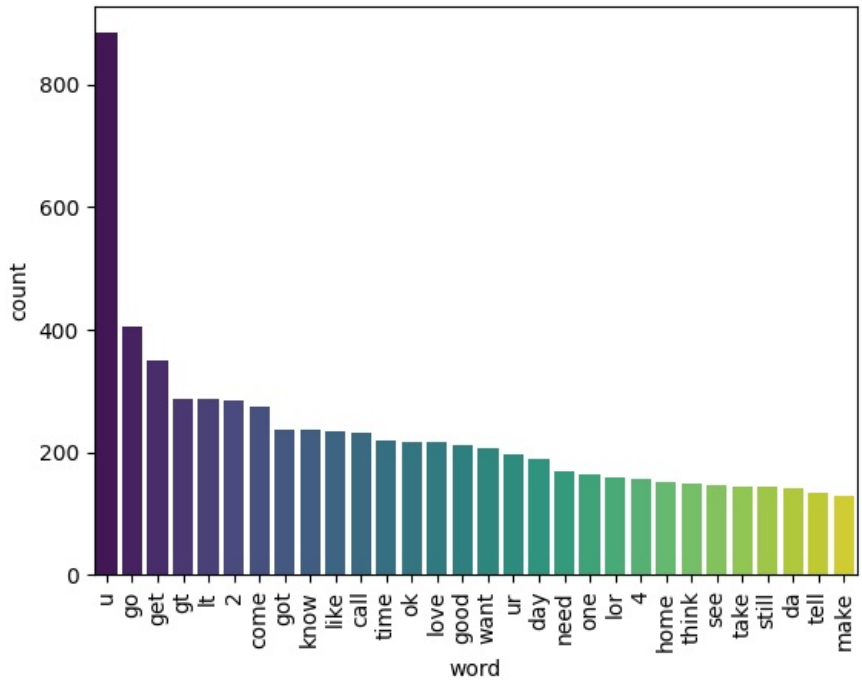


In [173…
```
#let's write the code and understand most frequently occurance 30 ham word or messages which means not spam
ham_curpus=[]
for msg in df[df['target']==0]['transfrom_text'].tolist():
    for word in msg.split():
        ham_curpus.append(word)
```

In [174…
```
len(ham_curpus)
```

Out[174]:
```
35404
```

In [175…
```
# Create a DataFrame for the most common 30 words
word_counts = pd.DataFrame(Counter(ham_curpus).most_common(30), columns=['word', 'count'])
# Plot the barplot using x and y keyword arguments
sns.barplot(x='word', y='count', data=word_counts,palette='viridis')
# Rotate the x-axis labels for better readability
plt.xticks(rotation='vertical')
# Show the plot
plt.show()
```



In [176…
```
# Text Vectorization
# using Bag of Words
df.head()
```

Out[176]:

	target	text	num_char	num_words	num_sentenc	transfrom_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

5. Model Building

In [177…
```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
cv=CountVectorizer()
tfidf = TfidfVectorizer(max_features=3000)
```

In [178…
```
x=tfidf.fit_transform(df['transfrom_text']).toarray()
```

In [179…
```
x.shape
```

Out[179]:
```
(5169, 3000)
```

In [180…
```
y=df['target'].values
y
```

Out[180]:
```
array([0, 0, 1, ..., 0, 0, 0])
```

In [181…
```
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)
```

In [182…
```
from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
```

In [183…
```
Gauss_nb = GaussianNB()
Multi_nb = MultinomialNB()
Bernoul_nb = BernoulliNB()
```

In [184…
```
Gauss_nb.fit(x_train,y_train)
y_pred1 = Gauss_nb.predict(x_test)
print(accuracy_score(y_test,y_pred1))
print(confusion_matrix(y_test,y_pred1))
print(precision_score(y_test,y_pred1))
```

```
0.874274661508704
[[790 106]
 [ 24 114]]
0.5181818181818182
```

In [185…
```
Multi_nb.fit(x_train,y_train)
```

```
y_pred2 = Multi_nb.predict(x_test)
print(accuracy_score(y_test,y_pred2))
print(confusion_matrix(y_test,y_pred2))
print(precision_score(y_test,y_pred2))
```

```
0.9709864603481625
[[896   0]
 [ 30 108]]
1.0
```

In [186…
```
Bernoul_nb.fit(x_train,y_train)
y_pred3 = Bernoul_nb.predict(x_test)
print(accuracy_score(y_test,y_pred3))
print(confusion_matrix(y_test,y_pred3))
print(precision_score(y_test,y_pred3))
```

```
0.9835589941972921
[[895   1]
 [ 16 122]]
0.991869918699187
```

In [187…
```
# in this time we will go of tfidf
```

In [188…
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
```

In [189…
```
svc = SVC(kernel='sigmoid', gamma=1.0)
knc = KNeighborsClassifier()
mnb = MultinomialNB()
dtc = DecisionTreeClassifier(max_depth=5)
lrc = LogisticRegression(solver='liblinear', penalty='l1')
rfc = RandomForestClassifier(n_estimators=50, random_state=2)
abc = AdaBoostClassifier(n_estimators=50, random_state=2)
bc = BaggingClassifier(n_estimators=50, random_state=2)
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
gbdt = GradientBoostingClassifier(n_estimators=50,random_state=2)
xgb = XGBClassifier(n_estimators=50,random_state=2)
```

In [190…
```
clfs = {
    'SVC' : svc,
    'KN' : knc,
    'NB': mnb,
    'DT': dtc,
    'LR': lrc,
    'RF': rfc,
    'AdaBoost': abc,
    'BgC': bc,
    'ETC': etc,
    'GBDT':gbdt,
    'xgb':xgb
}
```

In [191…
```
def train_classifier(clf,x_train,y_train,x_test,y_test):
    clf.fit(x_train,y_train)
    y_pred = clf.predict(x_test)
    accuracy = accuracy_score(y_test,y_pred)
    precision = precision_score(y_test,y_pred)

    return accuracy,precision
```

In [192…
```
print(train_classifier(svc,x_train,y_train,x_test,y_test))#svc stand for support vector classifier
```

```
(0.9758220502901354, np.float64(0.9747899159663865))
```

We will loop through the clfs dictionary, apply each algorithm to train the model, and store the accuracy and precision scores for every algorithm. Finally, we will convert the results into a new DataFrame

In [193…
```
accuracy_scores = []
precision_scores = []

for name,clf in clfs.items():

    current_accuracy,current_precision = train_classifier(clf, x_train, y_train, x_test, y_test)

    print("For ",name)
    print("Accuracy - ",current_accuracy)
    print("Precision - ",current_precision)
```

```
    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
```

```
For  SVC
Accuracy -  0.9758220502901354
Precision -  0.9747899159663865
For  KN
Accuracy -  0.9052224371373307
Precision -  1.0
For  NB
Accuracy -  0.9709864603481625
Precision -  1.0
For  DT
Accuracy -  0.9332688588007737
Precision -  0.8415841584158416
For  LR
Accuracy -  0.9555125725338491
Precision -  0.96
For  RF
Accuracy -  0.9738878143133463
Precision -  0.9826086956521739
For  AdaBoost
Accuracy -  0.9690522243713733
Precision -  0.9732142857142857
For  BgC
Accuracy -  0.9584139264990329
Precision -  0.8682170542635659
For  ETC
Accuracy -  0.9748549323017408
Precision -  0.9745762711864406
For  GBDT
Accuracy -  0.9506769825918762
Precision -  0.9306930693069307
For  xgb
Accuracy -  0.965183752417795
Precision -  0.9396551724137931
```

In [194…
```
df_performance = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy':accuracy_scores,'Precision':precision_scores})
df_performance
```

Out[194]:

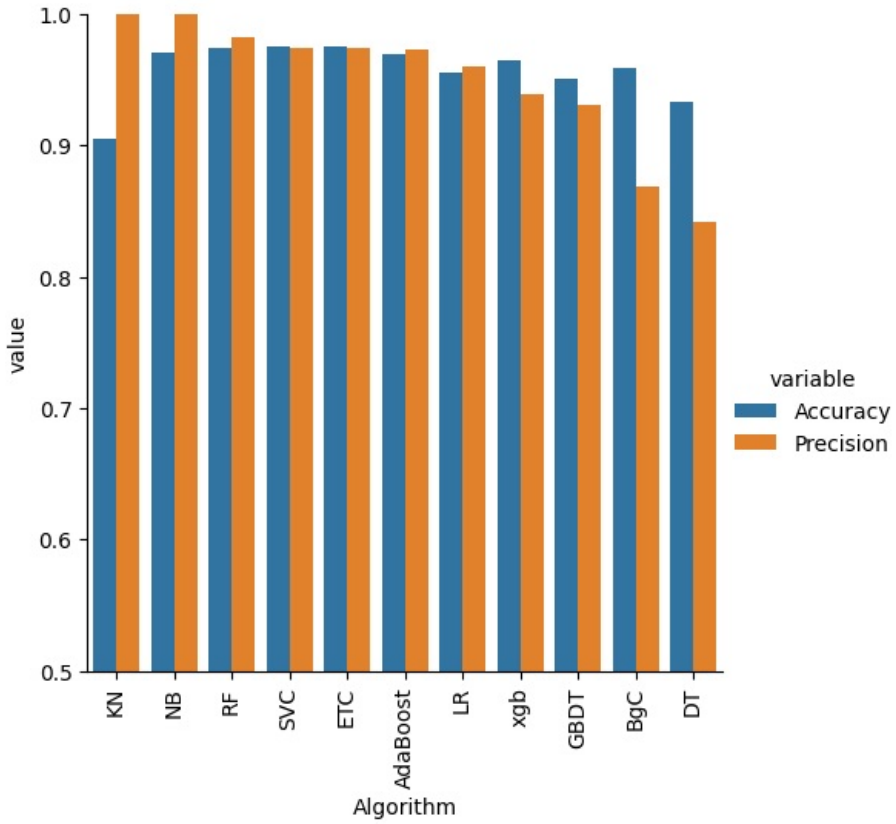
	Algorithm	Accuracy	Precision
1	KN	0.905222	1.000000
2	NB	0.970986	1.000000
5	RF	0.973888	0.982609
0	SVC	0.975822	0.974790
8	ETC	0.974855	0.974576
6	AdaBoost	0.969052	0.973214
4	LR	0.955513	0.960000
10	xgb	0.965184	0.939655
9	GBDT	0.950677	0.930693
7	BgC	0.958414	0.868217
3	DT	0.933269	0.841584

In [195…
```
df1_performance = pd.melt(df_performance, id_vars = "Algorithm")
df1_performance
```

Out[195]:

	Algorithm	variable	value
0	KN	Accuracy	0.905222
1	NB	Accuracy	0.970986
2	RF	Accuracy	0.973888
3	SVC	Accuracy	0.975822
4	ETC	Accuracy	0.974855
5	AdaBoost	Accuracy	0.969052
6	LR	Accuracy	0.955513
7	xgb	Accuracy	0.965184
8	GBDT	Accuracy	0.950677
9	BgC	Accuracy	0.958414
10	DT	Accuracy	0.933269
11	KN	Precision	1.000000
12	NB	Precision	1.000000
13	RF	Precision	0.982609
14	SVC	Precision	0.974790
15	ETC	Precision	0.974576
16	AdaBoost	Precision	0.973214
17	LR	Precision	0.960000
18	xgb	Precision	0.939655
19	GBDT	Precision	0.930693
20	BgC	Precision	0.868217
21	DT	Precision	0.841584

In [196…
```
sns.catplot(x = 'Algorithm', y='value', hue = 'variable',data=df1_performance, kind='bar',height=5)
plt.ylim(0.5,1.0)
plt.xticks(rotation='vertical')
plt.show()
```



In [197…
```
# model improve
# 1. Change the max_features parameter of TfIdf
```

In [198…
```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_max_ft_3000':accuracy_scores,'Precision_max_ft_3000':
```

In [199…
```
# temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_scaling':accuracy_scores,'Precision_scaling':precis
```

In [200…
```
# new_df = df_performance.merge(temp_df,on='Algorithm')
```

In [201…
```
new_df_scaled = new_df.merge(temp_df,on='Algorithm')
```

In [202…
```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_num_chars':accuracy_scores,'Precision_num_chars':prec
```

In [203…
```
new_df_scaled.merge(temp_df,on='Algorithm')
```

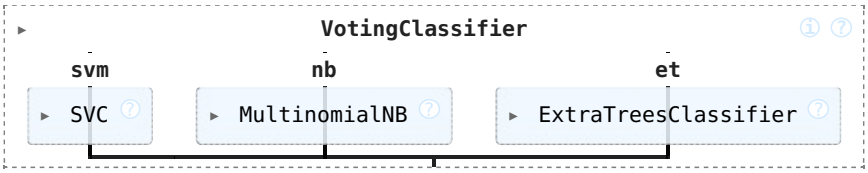
Out[203]:

	Algorithm	Accuracy	Precision	Accuracy_scaling	Precision_scaling	Accuracy_max_ft_3000	Precision_max_ft_3000	Accuracy_num_chars
0	KN	0.905222	1.000000	0.905222	1.000000	0.905222	1.000000	0.90522
1	NB	0.970986	1.000000	0.970986	1.000000	0.970986	1.000000	0.97098
2	RF	0.973888	0.982609	0.973888	0.982609	0.973888	0.982609	0.97388
3	SVC	0.975822	0.974790	0.975822	0.974790	0.975822	0.974790	0.97582
4	ETC	0.974855	0.974576	0.974855	0.974576	0.974855	0.974576	0.97485
5	AdaBoost	0.969052	0.973214	0.969052	0.973214	0.969052	0.973214	0.96905
6	LR	0.955513	0.960000	0.955513	0.960000	0.955513	0.960000	0.95551
7	xgb	0.965184	0.939655	0.965184	0.939655	0.965184	0.939655	0.96518
8	GBDT	0.950677	0.930693	0.950677	0.930693	0.950677	0.930693	0.95067
9	BgC	0.958414	0.868217	0.958414	0.868217	0.958414	0.868217	0.95841
10	DT	0.931335	0.838384	0.931335	0.838384	0.933269	0.841584	0.93326

In [204…
```
# Voting Classifier
svc = SVC(kernel='sigmoid', gamma=1.0,probability=True)
mnb = MultinomialNB()
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
```

In [205…
```
from sklearn.ensemble import VotingClassifier
voting = VotingClassifier(estimators=[('svm', svc), ('nb', mnb), ('et', etc)],voting='soft')
voting.fit(x_train,y_train)
```

Out[205]:



In [206…
```
y_pred = voting.predict(x_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

```
Accuracy 0.9796905222437138
Precision 0.9834710743801653
```

In [207…
```
# Applying stacking
estimators=[('svm', svc), ('nb', mnb), ('et', etc)]
final_estimator=RandomForestClassifier()
```

In [208…
```
from sklearn.ensemble import StackingClassifier
clf = StackingClassifier(estimators=estimators, final_estimator=final_estimator)
```

In [209…
```
clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

```
Accuracy 0.9787234042553191
Precision 0.9393939393939394
```

In [210…
```
import pickle
pickle.dump(tfidf,open('vectorizer.pkl','wb'))#wb stand for write binary
pickle.dump(mnb,open('model.pkl','wb'))
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: