EMAIL/SMS SPAM CLASSIFIER

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

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

In [3]:

```
email_data.head(15)
```

Out[3]:

	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
10	ham	I'm gonna be home soon and i don't want to tal...	NaN	NaN	NaN
11	spam	SIX chances to win CASH! From 100 to 20,000 po...	NaN	NaN	NaN
12	spam	URGENT! You have won a 1 week FREE membership ...	NaN	NaN	NaN
13	ham	I've been searching for the right words to tha...	NaN	NaN	NaN
14	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	NaN	NaN	NaN

In [4]:

```
email_data.dtypes
```

Out[4]:

```
v1            object
v2            object
Unnamed: 2    object
Unnamed: 3    object
Unnamed: 4    object
dtype: object
```

In [5]:

```
email_data.shape
```

Out[5]:

```
(5572, 5)
```

```
# we will be doing this project in following steps:
1.Data cleaning
2.EDA
3.Text Preprocessing
4.Model Building
5.Evaluation
6.Improvement
7.Website
8.Deploy
```

1.Data Cleaning

In [6]:

```
email_data.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 [7]:
```
email_data.isnull().sum()
```

Out[7]:
```
v1               0
v2               0
Unnamed: 2    5522
Unnamed: 3    5560
Unnamed: 4    5566
dtype: int64
```

In the last 3 columns most of the values are unknown so we will drop these (less than 5% are known)

In [8]:
```
email_data.columns
```

Out[8]:
```
Index(['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], dtype='object')
```

In [9]:
```
email_data.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], inplace=True)
```

In [10]:
```
#renaming the cols
email_data.rename(columns={"v1":"target", "v2":"text"}, inplace=True)
email_data.sample(5)
```

Out[10]:

	target	text
3086	ham	So i asked how's anthony. Dad. And your bf
5024	ham	I was gonna ask you lol but i think its at 7
2283	ham	I reach home safe n sound liao...
47	ham	Fair enough, anything going on?
2826	ham	Oh right, ok. I'll make sure that i do loads o...

In [11]:
```
email_data.loc[3859,:]
```

Out[11]:
```
target                                       ham
text      Yep. I do like the pink furniture tho.
Name: 3859, dtype: object
```

In [12]:
```
email_data.loc[88,:]
```

Out[12]:
```
target                                     ham
text      I'm really not up to it still tonight babe
Name: 88, dtype: object
```

In [13]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
```

In [14]:
```
encoder.fit_transform(email_data["target"])
```

Out[14]:
```
array([0, 0, 1, ..., 0, 0, 0])
```

In [15]:
```
email_data["target"] = encoder.fit_transform(email_data["target"])
```

In [16]:
```
email_data.sample(5)
```

Out[16]:

	target	text
984	0	Yo guess what I just dropped
4896	0	I cant pick the phone right now. Pls send a me...
3987	0	Hello. Sort of out in town already. That . So ...
2346	0	Its posible dnt live in &lt;#&gt; century cm ...
4823	0	Not thought bout it... || Drink in tap & spile...

In [17]:
```
email_data.isnull().sum()
```

Out[17]:
```
target    0
text      0
dtype: int64
```

In [18]:
```
#check for duplicate valeus
email_data.duplicated().sum()
```

Out[18]:
```
403
```

We will remove the duplicate rows

In [19]:
```
email_data.drop_duplicates()
```

Out[19]:

	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...
...	...	...
5567	1	This is the 2nd time we have tried 2 contact u...
5568	0	Will Ì_ b going to esplanade fr home?
5569	0	Pity, * was in mood for that. So...any other s...
5570	0	The guy did some bitching but I acted like i'd...
5571	0	Rofl. Its true to its name

5169 rows × 2 columns

In [20]:
```
email_data = email_data.drop_duplicates()
```

In [21]:
```
email_data.duplicated().sum()
```

Out[21]:
```
0
```

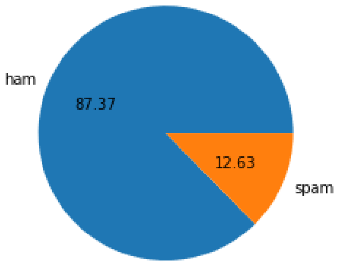
In [22]:
```
email_data.shape
```

Out[22]:
```
(5169, 2)
```

In [23]:
```
email_data["target"].value_counts()
```

Out[23]:
```
0    4516
1     653
Name: target, dtype: int64
```

In [24]:
```
import matplotlib.pyplot as plt
plt.pie(email_data["target"].value_counts(), labels=["ham","spam"], autopct="%0.2f")
plt.show()
```



Here the amount of spam is very less comparative to ham. We have unbalanced data

Now we will start the deeper analysis furthur

In [25]:
```
import nltk #natural library tool kit
```

In [26]:
```
nltk.download("punkt")
#nltk.download("punkt")is command used to download the Punkt tokenizer models,which are pre-trained models for tokenizing te
```

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

Out[26]:
```
True
```

In [27]:

```
# New Column: num of characters
email_data["text"].apply(len)
```

Out[27]:

```
0       111
1        29
2       155
3        49
4        61
       ...
5567    161
5568     37
5569     57
5570    125
5571     26
Name: text, Length: 5169, dtype: int64
```

In [28]:

```
email_data["num_characters"] = email_data["text"].apply(len)
```

In [29]:

```
email_data.sample(5)
```

Out[29]:

	target	text	num_characters
2210	0	Just wanted to say holy shit you guys weren't ...	68
2487	0	K ill drink.pa then what doing. I need srs mod...	78
2228	0	Those were my exact intentions	30
2006	0	Shopping lor. Them raining mah hard 2 leave or...	52
2414	0	O was not into fps then.	24

In [30]:

```
# New Column: num of words
email_data["text"].apply(lambda x:nltk.word_tokenize(x))
```

Out[30]:

```
0       [Go, until, jurong, point, ,, crazy, .., Avail...
1                [Ok, lar, ..., Joking, wif, u, oni, ...]
2       [Free, entry, in, 2, a, wkly, comp, to, win, F...
3       [U, dun, say, so, early, hor, ..., U, c, alrea...
4       [Nah, I, do, n't, think, he, goes, to, usf, ,,...
                              ...
5567    [This, is, the, 2nd, time, we, have, tried, 2,...
5568     [Will, Ì_, b, going, to, esplanade, fr, home, ?]
5569    [Pity, ,, *, was, in, mood, for, that, ., So, ...
5570    [The, guy, did, some, bitching, but, I, acted,...
5571                  [Rofl, ., Its, true, to, its, name]
Name: text, Length: 5169, dtype: object
```

In [31]:

```
email_data["text"].apply(lambda x:len(nltk.word_tokenize(x)))
```

Out[31]:

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

In [32]:

```
# New Column: num of words
email_data["num_words"] = email_data["text"].apply(lambda x:len(nltk.word_tokenize(x)))
```

In [33]:

```
email_data.sample(5)
```

Out[33]:

	target	text	num_characters	num_words
2184	0	I know a few people I can hit up and fuck to t...	52	14
3941	0	She's borderline but yeah whatever.	35	7
5152	0	Idk. I'm sitting here in a stop and shop parki...	184	43
466	0	They don't put that stuff on the roads to keep...	83	18
395	0	From here after The performance award is calcu...	102	17

In [34]:

```
#New Column: Num Of Sentences
email_data["num_sentences"] = email_data["text"].apply(lambda x:len(nltk.sent_tokenize(x)))
```

In [35]:
```
email_data.head(5)
```

Out[35]:

	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

In [36]:
```
email_data.columns
```

Out[36]:
```
Index(['target', 'text', 'num_characters', 'num_words', 'num_sentences'], dtype='object')
```

In [38]:
```
#for ham messages
email_data[email_data["target"]==0][['num_characters', 'num_words', 'num_sentences']].describe()
```

Out[38]:

	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

In [39]:
```
email_data[email_data["target"]==1][['num_characters', 'num_words', 'num_sentences']].describe()
```

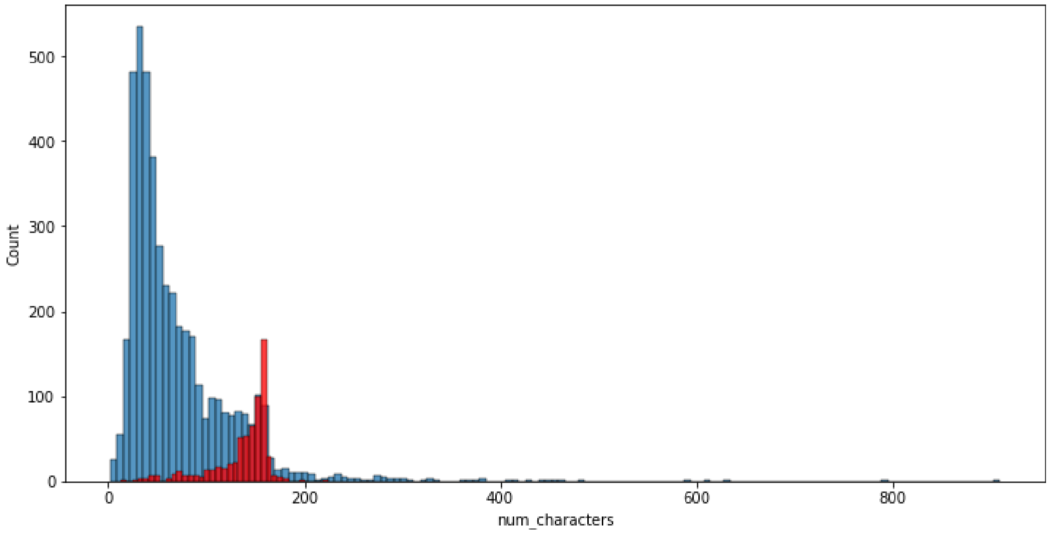
Out[39]:

	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

```
we can observe the spam messages are comparatively longer than the ham messages
mean num_characters for Ham < mean num_characters for Spam
```

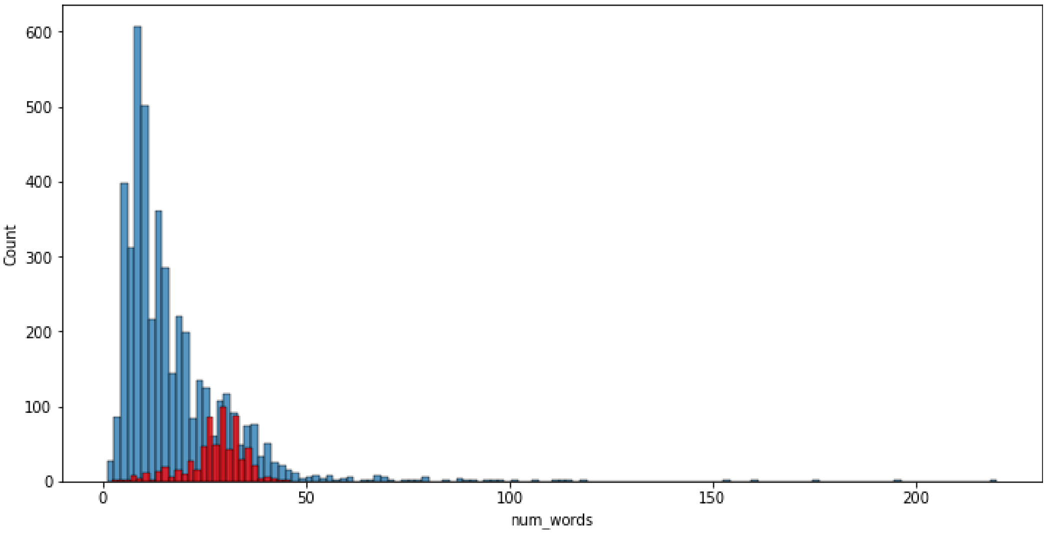
In [40]:
```
plt.figure(figsize=(12,6))
sns.histplot(email_data[email_data["target"]==0]["num_characters"])
sns.histplot(email_data[email_data["target"]==1]["num_characters"],color="red")
```

Out[40]:
```
<AxesSubplot:xlabel='num_characters', ylabel='Count'>
```



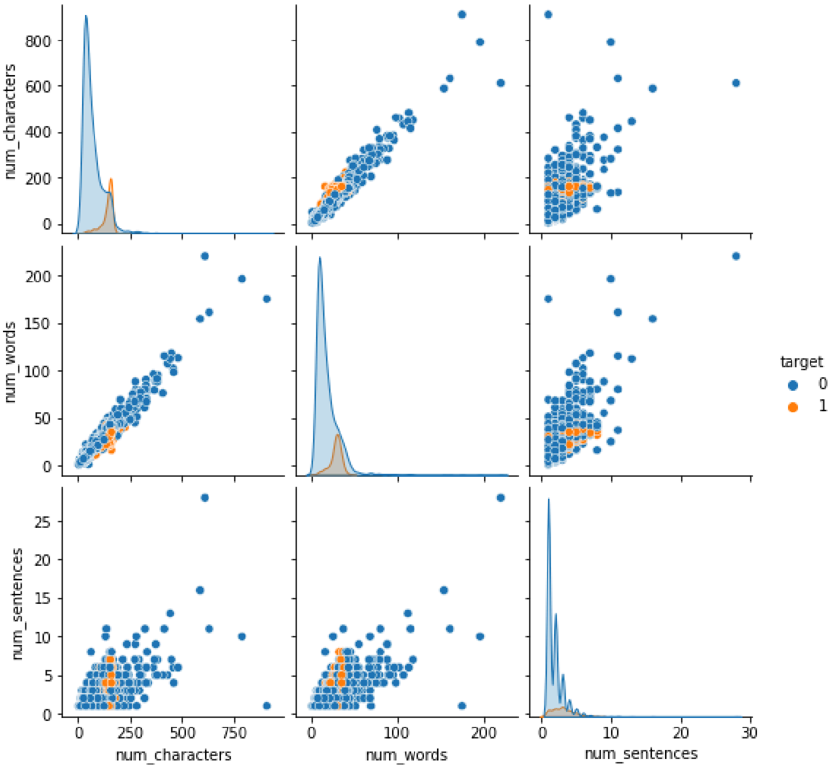
```
In [41]: plt.figure(figsize=(12,6))
         sns.histplot(email_data[email_data["target"]==0]["num_words"])
         sns.histplot(email_data[email_data["target"]==1]["num_words"],color="red")
```

```
Out[41]: <AxesSubplot:xlabel='num_words', ylabel='Count'>
```



```
In [42]: sns.pairplot(email_data, hue="target")
```

```
Out[42]: <seaborn.axisgrid.PairGrid at 0x23096f4ac10>
```



```
there are outliers in ham messages#
```

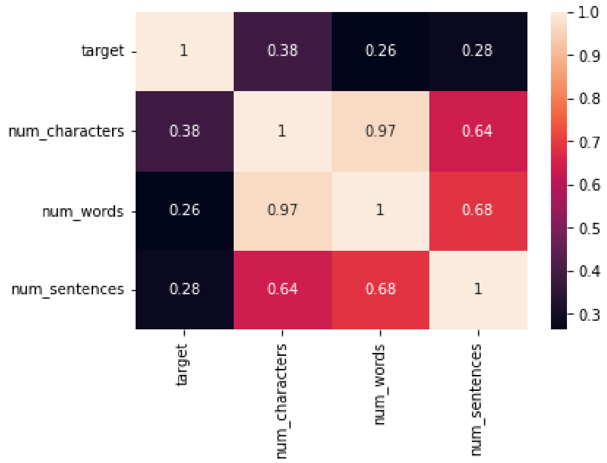
```
In [43]: email_data.corr()
```

Out[43]:

	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

In [44]:
```
sns.heatmap(email_data.corr(), annot=True)
```

Out[44]:
```
<AxesSubplot:>
```



There is multicollinearity in this dataset as high correlation between num_characters and num_words. Here, We will keep one of these 3 columns. we will keep only num_characters only as having 0.38 higher correlation.

3. Data Preprocessing

Lower Case(converting in lower case to all mail)
Tokenization(breaking in words)
Removing special characters(removing %, $,_u etc as no importance in meaning)
Removing stop words and punctuation(removing those words which are required in sentence formation but no meaning)
stemming(also called limitization,removes those words which are repeated as they having same meaning)

In [45]:
```
from nltk.corpus import stopwords
stopwords.words('english')
```

```
 'aren',
 "aren't",
 'as',
 'at',
 'be',
 'because',
 'been',
 'before',
 'being',
 'below',
 'between',
 'both',
 'but',
 'by',
 'can',
 'couldn',
 "couldn't",
 'd',
 'did',
```

In [46]:
```
import nltk
nltk.download('stopwords')
```

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

Out[46]:
```
True
```

In [47]:
```
import string
string.punctuation
```

Out[47]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [48]:
```
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
ps.stem("dancing")
```

Out[48]:
```
'danc'
```

In [49]:
```
def transform_text(text):#####
    text = text.lower()
    text = nltk.word_tokenize(text)
    y =[]
    for i in text:
        if i.isalnum():
            y.append(i)
    txt = y[:]
    y.clear()
    for i in txt:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)
    txt2 = y[:]
    y.clear()
    for i in txt2:
        y.append(ps.stem(i))
    return " ".join(y)
```

In [50]:
```
email_data["text"].apply(transform_text)
```

Out[50]:
```
0       go jurong point crazi avail bugi n great world...
1                                    ok lar joke wif u oni
2       free entri 2 wkli comp win fa cup final tkt 21...
3                      u dun say earli hor u c alreadi say
4                     nah think goe usf live around though
                              ...
5567    2nd time tri 2 contact u pound prize 2 claim e...
5568                                 b go esplanad fr home
5569                                     piti mood suggest
5570    guy bitch act like interest buy someth els nex...
5571                                        rofl true name
Name: text, Length: 5169, dtype: object
```

In [51]:
```
transform_text(email_data["text"].iloc[1])
```

Out[51]:
```
'ok lar joke wif u oni'
```

In [52]:
```
email_data["text"].iloc[1]
```

Out[52]:
```
'Ok lar... Joking wif u oni...'
```

In [53]:
```
email_data.loc[1, "text"]
```

Out[53]:
```
'Ok lar... Joking wif u oni...'
```

In [54]:
```
transform_text(email_data["text"].iloc[1])
```

Out[54]:
```
'ok lar joke wif u oni'
```

In [55]:
```
email_data["transformed_text"] = email_data["text"].apply(transform_text)
```

```
Now, We will make Word Cloud
```

In [56]:
```
from wordcloud import WordCloud
wc = WordCloud(width=500, height=500,min_font_size=10,background_color="white")
```

In [57]:
```
#spam_wc = wc.generate(email_data[email_data["target"]==1]["transformed_text"].str.cat(sep= " "))
#plt.imshow(spam_wc)
```

In [58]:
```
#ham_wc = wc.generate(email_data[email_data["target"]==0]["transformed_text"].str.cat(sep= " "))
#plt.imshow(ham_wc)
```

In [ ]:
```
# Now we will check for top words in spam and ham
```

In [59]:
```
spam_words = []
for msg in email_data[email_data["target"]==1]["transformed_text"].tolist():
    for word in msg.split():
        spam_words.append(word)
```

In [60]:
```python
print(spam_words)
```

```
'shracomorsglsuplt', '10', 'ls1', '3aj', 'hear', 'new', 'come', 'ken', 'stuff', 'pleas', 'call', 'custom', 'servic', 're
pres', '0800', '169', '6031', 'guarante', 'cash', 'prize', 'free', 'rington', 'wait', 'collect', 'simpli', 'text', 'pass
word', '85069', 'verifi', 'get', 'usher', 'britney', 'fml', 'gent', 'tri', 'contact', 'last', 'weekend', 'draw', 'show',
'prize', 'guarante', 'call', 'claim', 'code', 'k52', 'valid', '12hr', '150ppm', 'winner', 'u', 'special', 'select', '2',
'receiv', '4', 'holiday', 'flight', 'inc', 'speak', 'live', 'oper', '2', 'claim', 'privat', '2004', 'account', 'statemen
t', '07742676969', 'show', '786', 'unredeem', 'bonu', 'point', 'claim', 'call', '08719180248', 'identifi', 'code', '4523
9', 'expir', 'urgent', 'mobil', 'award', 'bonu', 'caller', 'prize', 'final', 'tri', 'contact', 'u', 'call', 'landlin',
'09064019788', 'box42wr29c', '150ppm', 'today', 'voda', 'number', 'end', '7548', 'select', 'receiv', '350', 'award', 'ma
tch', 'pleas', 'call', '08712300220', 'quot', 'claim', 'code', '4041', 'standard', 'rate', 'app', 'sunshin', 'quiz', 'wk
li', 'q', 'win', 'top', 'soni', 'dvd', 'player', 'u', 'know', 'countri', 'algarv', 'txt', 'ansr', 'sp', 'tyron', 'want',
'2', 'get', 'laid', 'tonight', 'want', 'real', 'dog', 'locat', 'sent', 'direct', '2', 'ur', 'mob', 'join', 'uk', 'larges
t', 'dog', 'network', 'bt', 'txting', 'gravel', '69888', 'nt', 'ec2a', '150p', 'rcv', 'msg', 'chat', 'svc', 'free', 'har
dcor', 'servic', 'text', 'go', '69988', 'u', 'get', 'noth', 'u', 'must', 'age', 'verifi', 'yr', 'network', 'tri', 'freem
sg', 'repli', 'text', 'randi', 'sexi', 'femal', 'live', 'local', 'luv', 'hear', 'netcollex', 'ltd', '08700621170150p',
'per', 'msg', 'repli', 'stop', 'end', 'custom', 'servic', 'annonc', 'new', 'year', 'deliveri', 'wait', 'pleas', 'call',
'07046744435', 'arrang', 'deliveri', 'winner', 'u', 'special', 'select', '2', 'receiv', 'cash', '4', 'holiday', 'fligh
t', 'inc', 'speak', 'live', 'oper', '2', 'claim', '0871277810810', 'stop', 'bootydeli', 'invit', 'friend', 'repli', 'se
e', 'stop', 'send', 'stop', 'frnd', '62468', 'bangbab', 'ur', 'order', 'way', 'u', 'receiv', 'servic', 'msg', '2', 'down
load', 'ur', 'content', 'u', 'goto', 'wap', 'bangb', 'tv', 'ur', 'mobil', 'menu', 'urgent', 'tri', 'contact', 'last', 'w
eekend', 'draw', 'show', 'prize', 'guarante', 'call', 'claim', 'code', 's89', 'valid', '12hr', 'pleas', 'call', 'custo
```

In [61]:
```python
from collections import Counter
dict_count = Counter(spam_words)
```

In [62]:
```python
list_words = sorted(dict_count.items(), key = lambda item:item[1], reverse=True)
```

In [63]:
```python
top_50_spam_words = list_words[0:50]
```

In [64]:
```python
print(top_50_spam_words)
```

```
[('call', 320), ('free', 191), ('2', 155), ('txt', 141), ('text', 122), ('u', 119), ('ur', 119), ('mobil', 114), ('stop',
104), ('repli', 103), ('claim', 98), ('4', 97), ('prize', 82), ('get', 74), ('new', 64), ('servic', 64), ('tone', 63), ('s
end', 60), ('urgent', 57), ('nokia', 57), ('contact', 56), ('award', 55), ('phone', 52), ('cash', 51), ('pleas', 51), ('we
ek', 49), ('win', 48), ('c', 45), ('collect', 45), ('min', 45), ('custom', 42), ('messag', 42), ('guarante', 42), ('per',
41), ('chat', 38), ('tri', 37), ('msg', 35), ('draw', 35), ('number', 35), ('cs', 35), ('show', 33), ('today', 33), ('offe
r', 33), ('line', 33), ('go', 32), ('receiv', 31), ('want', 31), ('latest', 30), ('rington', 30), ('landlin', 30)]
```

In [65]:
```python
print([word for word, freq in top_50_spam_words])
```

```
['call', 'free', '2', 'txt', 'text', 'u', 'ur', 'mobil', 'stop', 'repli', 'claim', '4', 'prize', 'get', 'new', 'servic',
'tone', 'send', 'urgent', 'nokia', 'contact', 'award', 'phone', 'cash', 'pleas', 'week', 'win', 'c', 'collect', 'min', 'cu
stom', 'messag', 'guarante', 'per', 'chat', 'tri', 'msg', 'draw', 'number', 'cs', 'show', 'today', 'offer', 'line', 'go',
'receiv', 'want', 'latest', 'rington', 'landlin']
```

In [66]:
```python
ham_words = []
for msg in email_data[email_data["target"]==0]["transformed_text"].tolist():
    for word in msg.split():
        ham_words.append(word)
```

In [67]:
```python
print(ham_words)
```

```
aw', 'class', 'gram', 'usual', 'run', 'like', 'lt', 'gt', 'half', 'eighth', 'smarter', 'though', 'get', 'almost', 'whol
e', 'second', 'gram', 'lt', 'gt', 'k', 'fyi', 'x', 'ride', 'earli', 'tomorrow', 'morn', 'crash', 'place', 'tonight', 'wo
w', 'never', 'realiz', 'embarass', 'accomod', 'thought', 'like', 'sinc', 'best', 'could', 'alway', 'seem', 'happi', 'sor
ri', 'give', 'sorri', 'offer', 'sorri', 'room', 'embarass', 'know', 'mallika', 'sherawat', 'yesterday', 'find', 'lt', 'u
rl', 'gt', 'sorri', 'call', 'later', 'meet', 'tell', 'reach', 'ye', 'gauti', 'sehwag', 'odi', 'seri', 'gon', 'na', 'pic
k', '1', 'burger', 'way', 'home', 'ca', 'even', 'move', 'pain', 'kill', 'ha', 'ha', 'ha', 'good', 'joke', 'girl', 'situa
t', 'seeker', 'part', 'check', 'iq', 'sorri', 'roommat', 'took', 'forev', 'ok', 'come', 'ok', 'lar', 'doubl', 'check',
'wif', 'da', 'hair', 'dresser', 'alreadi', 'said', 'wun', 'cut', 'v', 'short', 'said', 'cut', 'look', 'nice', 'today',
'dedic', 'day', 'song', 'u', 'dedic', 'send', 'ur', 'valuabl', 'frnd', 'first', 'rpli', 'plane', 'give', 'month', 'end',
'wah', 'lucki', 'man', 'save', 'money', 'hee', 'finish', 'class', 'hi', 'babe', 'im', 'home', 'wan', 'na', 'someth', 'x
x', 'k', 'k', 'perform', 'u', 'call', 'wait', 'machan', 'call', 'free', 'that', 'cool', 'gentleman', 'treat', 'digniti',
'respect', 'like', 'peopl', 'much', 'shi', 'pa', 'oper', 'lt', 'gt', 'still', 'look', 'job', 'much', 'ta', 'earn', 'sorr
i', 'call', 'later', 'call', 'ah', 'ok', 'way', 'home', 'hi', 'hi', 'place', 'man', 'yup', 'next', 'stop', 'call', 'late
r', 'network', 'urgnt', 'sm', 'real', 'u', 'get', 'yo', 'need', '2', 'ticket', 'one', 'jacket', 'done', 'alreadi', 'us
e', 'multi', 'ye', 'start', 'send', 'request', 'make', 'pain', 'came', 'back', 'back', 'bed', 'doubl', 'coin', 'factor
i', 'got', 'ta', 'cash', 'nitro', 'realli', 'still', 'tonight', 'babe', 'ela', 'il', 'download', 'come', 'wen', 'ur', 'f
ree', 'yeah', 'stand', 'close', 'catch', 'someth', 'sorri', 'pain', 'ok', 'meet', 'anoth', 'night', 'spent', 'late', 'af
ternoon', 'casualti', 'mean', 'done', 'stuff42moro', 'includ', 'time', 'sheet', 'sorri', 'smile', 'pleasur', 'smile', 'p
ain', 'smile', 'troubl', 'pour', 'like', 'rain', 'smile', 'sum1', 'hurt', 'u', 'smile', 'becoz', 'someon', 'still', 'lov
e', 'see', 'u', 'smile', 'havent', 'plan', 'buy', 'later', 'check', 'alreadi', 'lido', 'got', '530', 'show', 'e', 'after
```

In [68]:
```python
from collections import Counter
dict_count_ham = Counter(ham_words)
```

In [69]:
```
print(dict_count_ham)
```

```
ve': 39, 'person': 39, 'everi': 39, 'quit': 39, 'lar': 38, 'pay': 38, 'may': 38, 'help': 37, 'that': 37, 'liao': 37, 'da
t': 37, 'shop': 37, 'bring': 37, 'wonder': 36, 'month': 36, 'hello': 36, 'girl': 36, 'end': 36, 'yo': 36, 'hous': 36, 'm
inut': 36, 'kiss': 36, 'rememb': 35, 'dinner': 35, 'room': 35, 'x': 35, 'best': 35, 'guess': 35, 'ju': 35, 'readi': 35,
'min': 35, 'man': 34, 'noth': 34, 'might': 34, 'shit': 34, 'mind': 34, 'earli': 33, 'anoth': 33, 'aight': 33, 'sir': 33,
'stay': 33, 'big': 33, 'actual': 33, 'put': 33, 'god': 33, 'probabl': 33, 'wont': 32, 'ah': 32, 'bed': 32, 'anyway': 32,
'heart': 32, 'boy': 32, 'book': 32, 'den': 32, 'name': 31, 'show': 31, 'chang': 31, 'babi': 31, 'dunno': 31, 'princess':
31, 'word': 30, 'leh': 30, 'face': 30, 'thanx': 30, 'wake': 30, 'enjoy': 30, 'dad': 30, 'left': 29, 'sweet': 29, 'run':
29, 'hear': 29, 'shall': 29, 'bad': 29, 'world': 28, 'forgot': 28, 'tmr': 28, 'didnt': 28, 'two': 28, 'ever': 28, 'sat':
28, 'wif': 27, 'bu': 27, 'weekend': 27, 'hurt': 27, 'school': 27, 'mail': 27, 'littl': 27, 'walk': 27, 'everyth': 27, 'g
oe': 26, 'though': 26, 'lesson': 26, 'pain': 26, 'afternoon': 26, 'movi': 26, 'abt': 26, 'okay': 26, 'test': 26, 'abl':
26, 'sound': 26, 'luv': 26, 'di': 26, 'offic': 26, 'live': 25, 'enough': 25, 'decid': 25, 'sinc': 25, 'birthday': 25, 'p
lay': 25, 'juz': 25, 'bath': 25, 'speak': 24, 'saw': 24, 'hair': 24, 'havent': 24, 'smoke': 24, 'wot': 24, 'made': 24,
'dude': 24, '5': 24, 'bore': 24, 'town': 24, 'half': 23, 'came': 23, 'els': 23, 'haf': 23, 'without': 23, 'til': 23, 'ha
v': 23, 'oso': 23, 'fun': 23, 'onlin': 23, 'lei': 23, 'ard': 22, 'alright': 22, 'real': 22, 'special': 22, 'food': 22,
'head': 22, 'beauti': 22, 'sch': 22, 'goin': 22, 'mom': 21, 'wo': 21, 'pa': 21, 'caus': 21, 'must': 21, 'open': 21, 'dre
am': 21, 'tv': 21, 'nite': 21, 'read': 21, 'drink': 21, 'tot': 21, 'togeth': 21, 'drop': 21, '6': 21, 'second': 20, 'yes
terday': 20, 'busi': 20, 'account': 20, 'studi': 20, 'si': 20, 'decim': 20, 'noe': 20, 'full': 20, 'chikku': 20, 'huh':
20, 'detail': 20, 'famili': 20, 'de': 20, 'away': 20, 'treat': 19, 'set': 19, 'aft': 19, 'till': 19, 'tomo': 19, 'answe
r': 19, 'awesom': 19, 'true': 19, 'trip': 19, 'post': 19, 'mum': 19, 'train': 19, 'rite': 19, '9': 18, 'part': 18, 'frn
d': 18, 'close': 18, 'old': 18, 'question': 18, 'believ': 18, 'plz': 18, 'reason': 18, 'gd': 18, 'neva': 18, 'sad': 18,
```

In [70]:
```
list_words_ham = sorted(dict_count_ham.items(), key = lambda item:item[1], reverse=True)
```

In [71]:
```
top_50_ham_words = list_words_ham[0:50]
```

In [72]:
```
print([word for word, freq in top_50_ham_words])
```

```
['u', 'go', 'get', 'gt', 'lt', '2', 'come', 'got', 'know', 'like', 'call', 'time', 'ok', 'love', 'good', 'want', 'ur', 'da
y', 'need', 'one', 'lor', '4', 'home', 'think', 'see', 'take', 'still', 'da', 'tell', 'make', 'say', 'back', 'today', 'hop
e', 'ask', 'sorri', 'n', 'send', 'r', 'work', 'dont', 'meet', 'hi', 'well', 'thing', 'wat', 'k', 'much', 'night', 'oh']
```

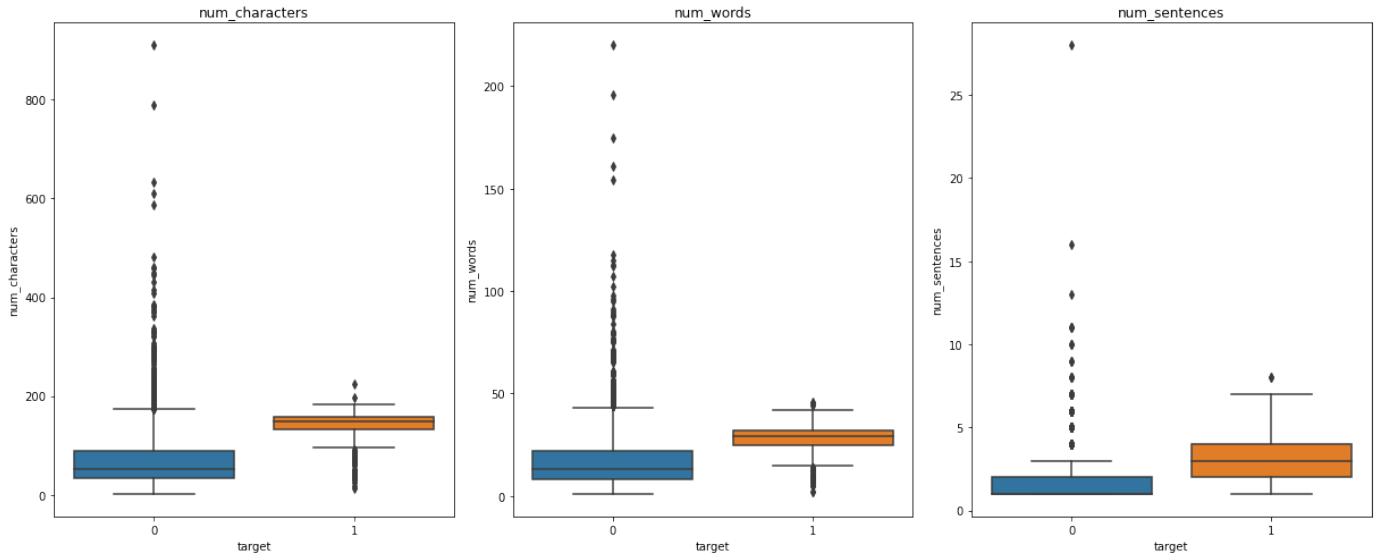
Hypothesis testing

Spam messages tend to be longer than ham messages in terms of character count, word count, and number of sentences.

In [73]:

```python
from scipy import stats
# Hypothesis 1: Length Hypothesis
def test_length_hypothesis(data):
    print("\n=== Testing Length Hypothesis ===")
    # Compare means
    for col in ['num_characters', 'num_words', 'num_sentences']:
        ham = data[data['target'] == 0][col]
        spam = data[data['target'] == 1][col]

        t_stat, p_val = stats.ttest_ind(spam, ham, equal_var=False)
        print(f"\n{col}:")
        print(f"Ham mean: {ham.mean():.2f}, Spam mean: {spam.mean():.2f}")
        print(f"T-test p-value: {p_val:.4f}")
        if p_val < 0.05:
            print("Significant difference - Hypothesis supported")
        else:
            print("No significant difference - Hypothesis not supported")
    plt.figure(figsize=(17, 7))
    for i, col in enumerate(['num_characters', 'num_words', 'num_sentences'], 1):
        plt.subplot(1, 3, i)
        sns.boxplot(x='target', y=col, data=data)
        plt.title(col)
    plt.tight_layout()
    plt.show()
test_length_hypothesis(email_data)
```

```
=== Testing Length Hypothesis ===

num_characters:
Ham mean: 70.46, Spam mean: 137.89
T-test p-value: 0.0000
Significant difference - Hypothesis supported

num_words:
Ham mean: 17.12, Spam mean: 27.67
T-test p-value: 0.0000
Significant difference - Hypothesis supported

num_sentences:
Ham mean: 1.80, Spam mean: 2.97
T-test p-value: 0.0000
Significant difference - Hypothesis supported
```



Certain words (like "free", "win", "prize", "call") appear more frequently in spam messages than in ham messages.

In [74]:
```
def test_word_frequency_hypothesis(data):
    print("\n=== Testing Word Frequency Hypothesis ===")
    from collections import Counter

    # Get top spam words
    spam_words = []
    for msg in data[data['target'] == 1]['transformed_text'].tolist():
        for word in msg.split():
            spam_words.append(word)

    spam_word_counts = Counter(spam_words)
    top_spam_words = [word for word, count in spam_word_counts.most_common(10)]

    # Get top ham words
    ham_words = []
    for msg in data[data['target'] == 0]['transformed_text'].tolist():
        for word in msg.split():
            ham_words.append(word)

    ham_word_counts = Counter(ham_words)
    top_ham_words = [word for word, count in ham_word_counts.most_common(10)]

    print("\nTop 10 Spam Words:", top_spam_words)
    print("Top 10 Ham Words:", top_ham_words)
test_word_frequency_hypothesis(email_data)
```

```
=== Testing Word Frequency Hypothesis ===

Top 10 Spam Words: ['call', 'free', '2', 'txt', 'text', 'u', 'ur', 'mobil', 'stop', 'repli']
Top 10 Ham Words: ['u', 'go', 'get', 'gt', 'lt', '2', 'come', 'got', 'know', 'like']
```

In [75]:
```
email_data["char_z"] = (email_data["num_characters"]-email_data["num_characters"].mean())/email_data["num_characters"].std()
```

In [76]:
```
email_data.head()
```

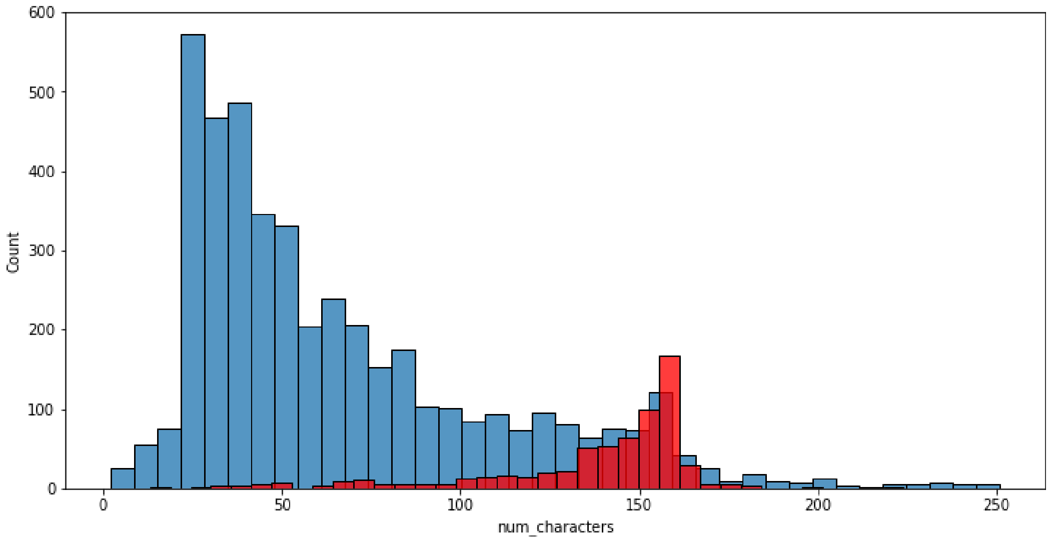
Out[76]:

	target	text	num_characters	num_words	num_sentences	transformed_text	char_z
0	0	Go until jurong point, crazy.. Available only ...	111	24	2	go jurong point crazi avail bugi n great world...	0.549864
1	0	Ok lar... Joking wif u oni...	29	8	2	ok lar joke wif u oni	-0.858192
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...	1.305407
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	-0.514764
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1	nah think goe usf live around though	-0.308707

In [77]:
```
email_data2 = email_data[(email_data["char_z"]<3) & (email_data["char_z"]>-3)]
```

In [78]:
```
plt.figure(figsize=(12,6))
sns.histplot(email_data2[email_data2["target"]==0]["num_characters"])
sns.histplot(email_data2[email_data2["target"]==1]["num_characters"],color="red")
```

Out[78]:
```
<AxesSubplot:xlabel='num_characters', ylabel='Count'>
```



Modelling

we need to convert the text into vector of numbers, there are varous method to do this. We will do it furthur

In [79]:
```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
```

In [80]:
```
X = cv.fit_transform(email_data["transformed_text"])
```

In [81]:
```
print(X)
```

```
  (0, 2762)	1
  (0, 3393)	1
  (0, 4608)	1
  (0, 1806)	1
  (0, 1017)	1
  (0, 1385)	1
  (0, 2835)	1
  (0, 6556)	1
  (0, 3497)	1
  (0, 1383)	1
  (0, 1610)	1
  (0, 2801)	1
  (0, 837)	1
  (0, 6391)	1
  (1, 4289)	1
  (1, 3528)	1
  (1, 3364)	1
  (1, 6484)	1
  (1, 4312)	1
  (2, 2610)	1
  (2, 2291)	2
  (2, 6524)	1
  (2, 1695)	1
  (2, 6494)	1
  (2, 2404)	2
  :	:
  (5164, 132)	1
  (5164, 4222)	1
  (5165, 2762)	1
  (5165, 3041)	1
  (5165, 2599)	1
  (5165, 2321)	1
  (5166, 5702)	1
  (5166, 3992)	1
  (5166, 4553)	1
  (5167, 2610)	1
  (5167, 6421)	1
  (5167, 3602)	1
  (5167, 4147)	1
  (5167, 6240)	1
  (5167, 5457)	1
  (5167, 1410)	1
  (5167, 2250)	1
  (5167, 2696)	1
  (5167, 2879)	1
  (5167, 3240)	1
  (5167, 705)	1
  (5167, 1215)	1
  (5168, 4077)	1
  (5168, 6099)	1
  (5168, 5038)	1
```

In [82]:
```
x = cv.fit_transform(email_data["transformed_text"]).toarray()
```

In [83]:
```
print(x)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

In [84]:
```
print(x.shape)
```

```
(5169, 6708)
```

In [85]:

```
y = email_data["target"]
print(y)
```

```
0       0
1       0
2       1
3       0
4       0
       ..
5567    1
5568    0
5569    0
5570    0
5571    0
Name: target, Length: 5169, dtype: int32
```

In [86]:

```
y_ = email_data["target"].values
print(y)
```

```
0       0
1       0
2       1
3       0
4       0
       ..
5567    1
5568    0
5569    0
5570    0
5571    0
Name: target, Length: 5169, dtype: int32
```

In [87]:

```
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=42)
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
```

In [88]:

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

In [89]:

```
gnb.fit(x_train,y_train)
```

Out[89]:

```
GaussianNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [90]:

```
y_pred1 = gnb.predict(x_test)
```

In [91]:

```
print(accuracy_score(y_test, y_pred1))
print(confusion_matrix(y_test, y_pred1))
print(precision_score(y_test, y_pred1))
```

```
0.8684719535783365
[[772 117]
 [ 19 126]]
0.5185185185185185
```

In [92]:

```
mnb.fit(x_train,y_train)
y_pred2 = mnb.predict(x_test)
print(accuracy_score(y_test, y_pred2))
print(confusion_matrix(y_test, y_pred2))
print(precision_score(y_test, y_pred2))
```

```
0.9738878143133463
[[872  17]
 [ 10 135]]
0.8881578947368421
```

In [93]:

```
bnb.fit(x_train,y_train)
y_pred3 = bnb.predict(x_test)
print(accuracy_score(y_test, y_pred3))
print(confusion_matrix(y_test, y_pred3))
print(precision_score(y_test, y_pred3))
```

```
0.9661508704061895
[[885   4]
 [ 31 114]]
0.9661016949152542
```

```
Other Algorithms
```

In [95]:
```
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
```

In [96]:
```
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)
```

In [97]:
```
clfs = {
    "SVC":svc,
    "KN":knc,
    "MNB":mnb,
    "GNB":gnb,
    "BNB":bnb,
    "DT":dtc,
    "LR":lrc,
    "RF":rfc,

}
```

In [98]:
```
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 [99]:
```
train_classifier(svc, x_train, y_train, x_test, y_test)
```

Out[99]:
```
(0.9332688588007737, 0.7676056338028169)
```

In [100]:
```
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 0.9332688588007737
Precision 0.7676056338028169
For KN
Accuracy 0.9042553191489362
Precision 1.0
For MNB
Accuracy 0.9738878143133463
Precision 0.8881578947368421
For GNB
Accuracy 0.8684719535783365
Precision 0.5185185185185185
For BNB
Accuracy 0.9661508704061895
Precision 0.9661016949152542
For DT
Accuracy 0.9235976789168279
Precision 0.9230769230769231
For LR
Accuracy 0.9709864603481625
Precision 0.9457364341085271
For RF
Accuracy 0.9661508704061895
Precision 1.0
```

In [101]:

```
print(accuracy_scores)
print(precision_scores)
```

```
[0.9332688588007737, 0.9042553191489362, 0.9738878143133463, 0.8684719535783365, 0.9661508704061895, 0.9235976789168279,
0.9709864603481625, 0.9661508704061895]
[0.7676056338028169, 1.0, 0.8881578947368421, 0.5185185185185185, 0.9661016949152542, 0.9230769230769231, 0.94573643410852
71, 1.0]
```

In [102]:

```
performance_df = pd.DataFrame({"Algorithm":clfs.keys(), "Accuracy":accuracy_scores, "Precision":precision_scores})
```

In [103]:

```
performance_df
```

Out[103]:

	Algorithm	Accuracy	Precision
0	SVC	0.933269	0.767606
1	KN	0.904255	1.000000
2	MNB	0.973888	0.888158
3	GNB	0.868472	0.518519
4	BNB	0.966151	0.966102
5	DT	0.923598	0.923077
6	LR	0.970986	0.945736
7	RF	0.966151	1.000000

Testing of MOdel

In [104]:

```
mnb.predict(x_test)[0:5]
```

Out[104]:

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

In [105]:

```
print(y_test)
```

```
1617    0
2064    0
1272    0
3020    0
3642    0
       ..
4146    0
1208    0
4795    1
3575    0
2820    0
Name: target, Length: 1034, dtype: int32
```

In [106]:

```
rfc.predict(x_test)[0:5]
```

Out[106]:

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

```
HERE BNB(BERNOULLI NAIVE BAYS) GIVES HIGHEST PRECISION AND SECOND HIGHEST ACCURACY.THIS ALGO WILL GIVE GOOD RESULTS FOR
OUR MODEL.AS OUR DATA IS IMBALANCED, SO WE SHOULD CONSIDER PRECISION OVER ACCURACY AS THE METRIC TO EVALUATE THE MODEL
PERFORMANCE.
```