sms_spam_detection

August 23, 2025

[1]:
```python
import numpy as np
import pandas as pd
```

[2]:
```python
import sys
print(sys.executable)
```

```
/Users/ayush/Desktop/spam-classifier/myenv/bin/python
```

[3]:
```python
df=pd.read_csv('spam.csv', encoding='latin1')
df.head(5)
```

[3]:
```
     v1                                                 v2 Unnamed: 2  \
0   ham  Go until jurong point, crazy.. Available only …        NaN
1   ham                      Ok lar… Joking wif u oni…        NaN
2  spam  Free entry in 2 a wkly comp to win FA Cup fina…        NaN
3   ham  U dun say so early hor… U c already then say…        NaN
4   ham  Nah I don't think he goes to usf, he lives aro…        NaN

  Unnamed: 3 Unnamed: 4
0        NaN        NaN
1        NaN        NaN
2        NaN        NaN
3        NaN        NaN
4        NaN        NaN
```

[4]:
```python
df.shape
```

[4]:
```
(5572, 5)
```

1.Data Cleaning 2.EDA 3.Text Preprocessing 4.Model Building 5.Evaluation 6.Improvement 7.Website 8.Deployment

[5]:
```python
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
```

[6]:
```
df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace=True)
```

[7]:
```
df.sample()
```

[7]:
```
      v1                                                 v2
429  ham  I wnt to buy a BMW car urgently..its vry urgen…
```

[8]:
```
df.rename(columns={'v1':'target','v2':'text'},inplace=True)
```

[9]:
```
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
```

[10]:
```
df['target']=encoder.fit_transform(df['target'])
```

[11]:
```
df.head(5)
```

[11]:
```
   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…
```

[12]:
```
df.isnull().sum()
```

[12]:
```
target    0
text      0
dtype: int64
```

[13]:
```
df.duplicated().sum()
```

[13]:
```
np.int64(403)
```

[14]:
```
df=df.drop_duplicates(keep='first')
```

[15]:
```
df.duplicated().sum()
```

[15]:
```
np.int64(0)
```

[16]:
```
df.shape
```

[16]:
```
(5169, 2)
```

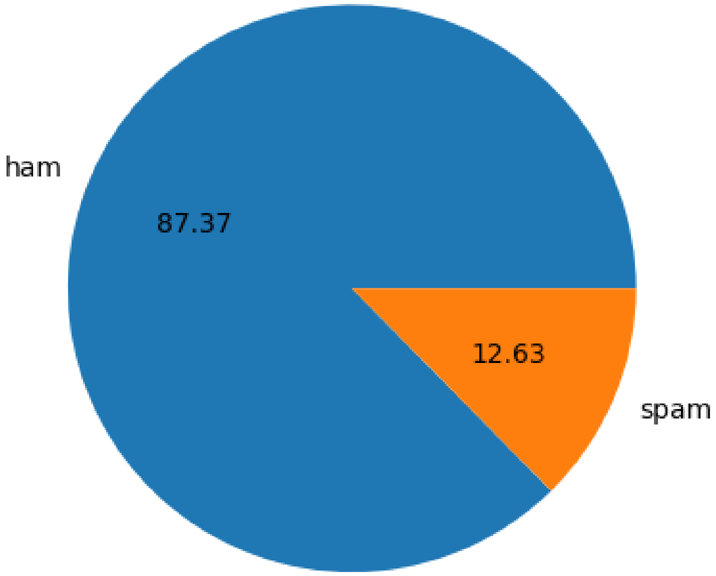
//EDA

[17]:
```
df['target'].value_counts()
```

[17]:
```
target
0    4516
1     653
Name: count, dtype: int64
```

[18]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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



[19]:
```
import nltk
```

[20]:
```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /Users/ayush/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[20]:
```
True
```

```
[21]: nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to /Users/ayush/nltk_data…
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
[21]: True
```

```
[22]: df['num_characters']=df['text'].apply(len)
```

```
[23]: df.head()
```

```
[23]:
   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
```

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

```
[24]:
   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

   num_words
0         24
1          8
2         37
3         13
4         15
```

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

```
[25]:
   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

   num_words  num_sentences
0         24              2
1          8              2
```

```
2          37               2
3          13               1
4          15               1
```

[26]:
```
df[['num_characters','num_words','num_sentences']].describe()
```

[26]:
```
       num_characters    num_words  num_sentences
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
```

[27]:
```
df[df['target']==0][['num_characters','num_words','num_sentences']].describe()
```

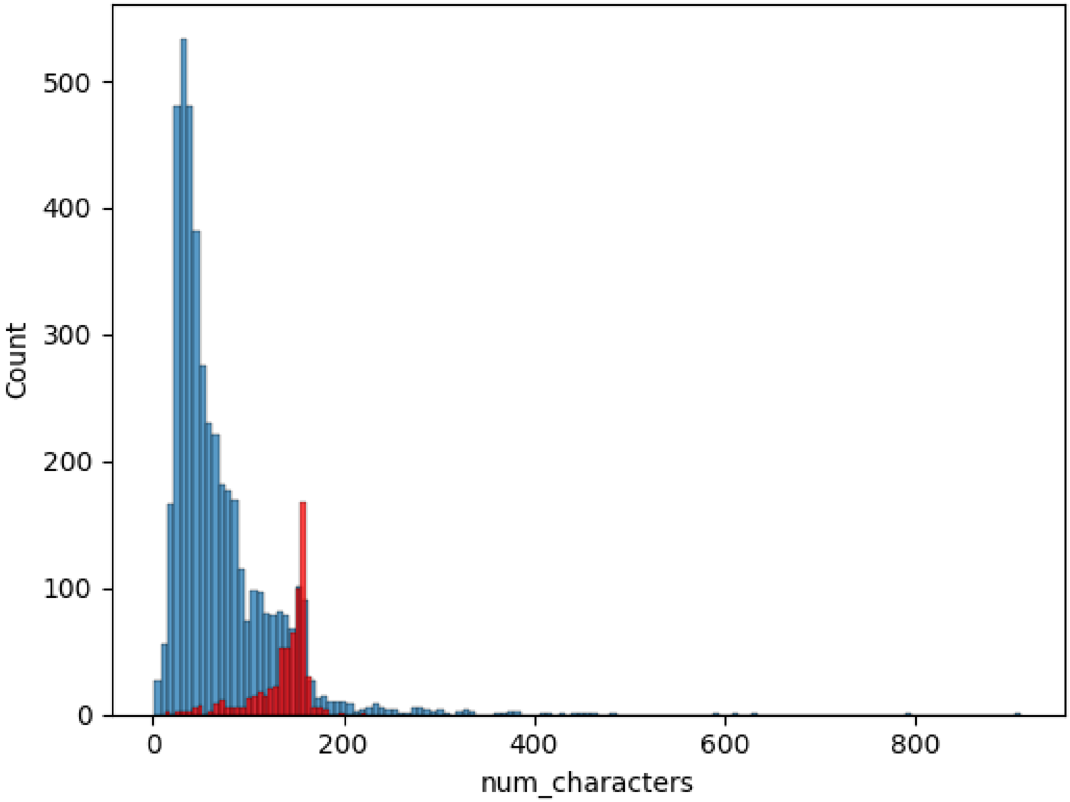
[27]:
```
       num_characters    num_words  num_sentences
count     4516.000000  4516.000000    4516.000000
mean        70.459256    17.123782       1.820195
std         56.358207    13.493970       1.383657
min          2.000000     1.000000       1.000000
25%         34.000000     8.000000       1.000000
50%         52.000000    13.000000       1.000000
75%         90.000000    22.000000       2.000000
max        910.000000   220.000000      38.000000
```

[28]:
```
df[df['target']==1][['num_characters','num_words','num_sentences']].describe()
```

[28]:
```
       num_characters   num_words  num_sentences
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
```

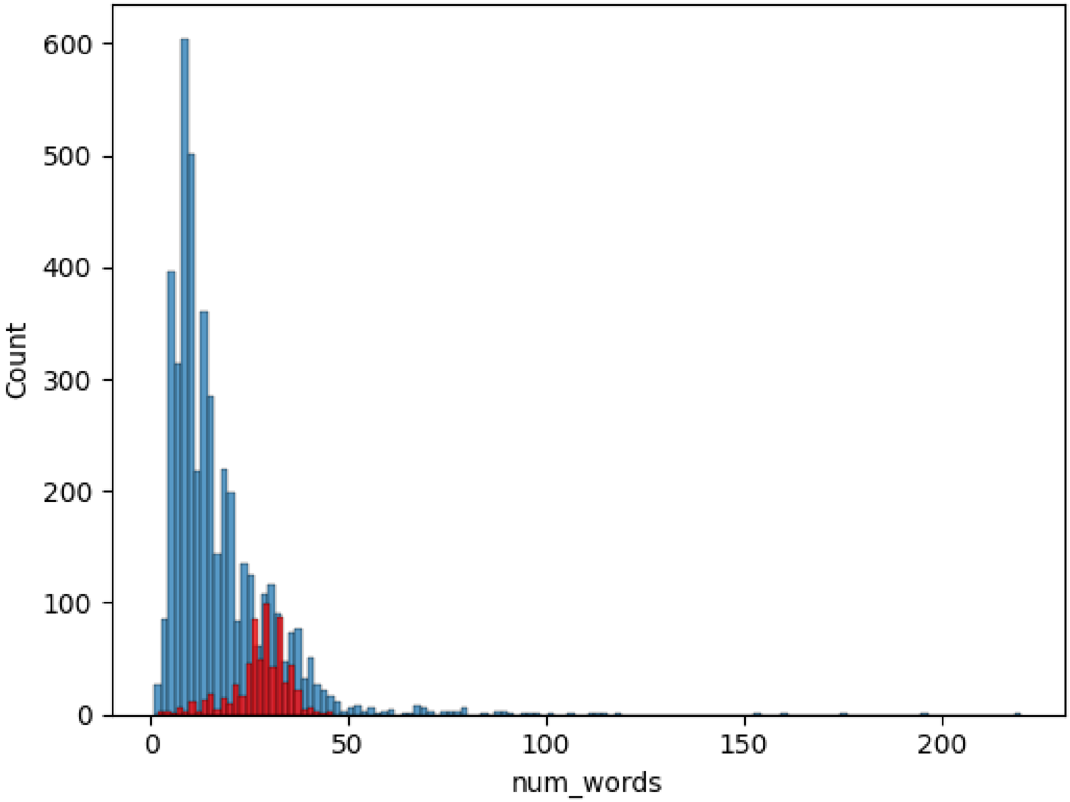
[29]:
```
sns.histplot(df[df['target']==0]['num_characters'])
sns.histplot(df[df['target']==1]['num_characters'],color='red')
```

[29]:
```
<Axes: xlabel='num_characters', ylabel='Count'>
```



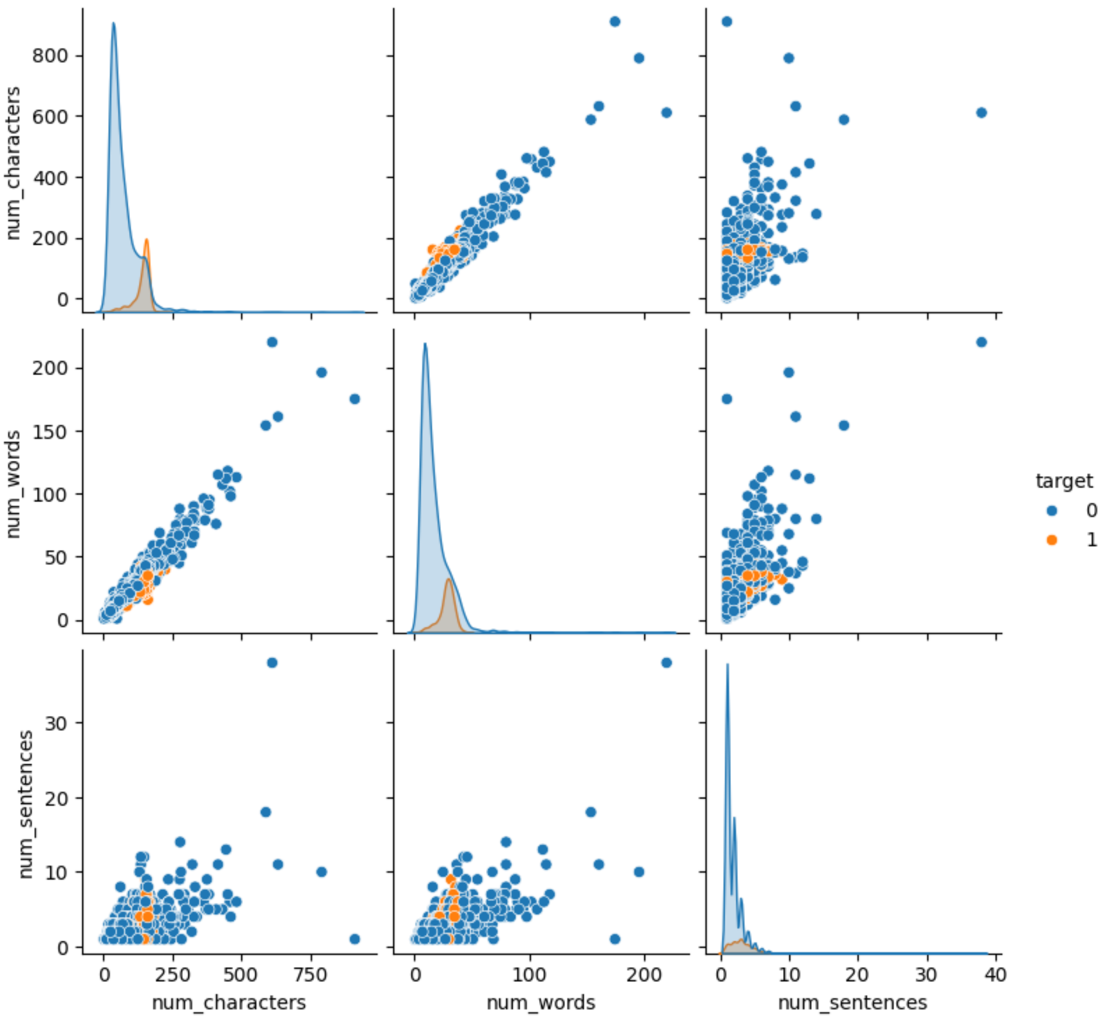
[30]:
```
sns.histplot(df[df['target']==0]['num_words'])
sns.histplot(df[df['target']==1]['num_words'],color='red')
```

[30]:
```
<Axes: xlabel='num_words', ylabel='Count'>
```



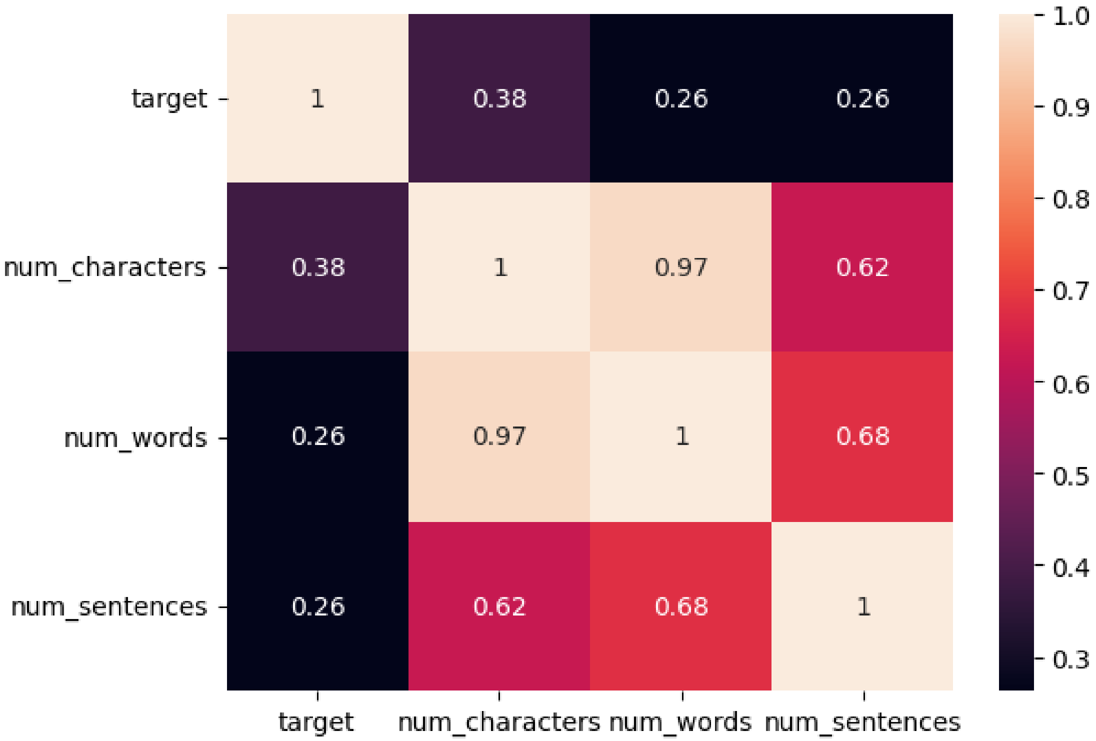
[31]:
```
sns.pairplot(df,hue='target')
```

[31]:
```
<seaborn.axisgrid.PairGrid at 0x14361ef10>
```



```
[32]: sns.heatmap(df.select_dtypes(include=['int64', 'float64']).corr(),annot=True)
```

```
[32]: <Axes: >
```



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

```
[nltk_data] Downloading package stopwords to /Users/ayush/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

```
[33]: True
```

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

```
[34]: ['a',
       'about',
       'above',
       'after',
       'again',
       'against',
       'ain',
       'all',
       'am',
       'an',
       'and',
       'any',
       'are',
       'aren',
       "aren't",
       'as',
       'at',
       'be',
       'because',
       'been',
       'before',
       'being',
       'below',
       'between',
       'both',
       'but',
       'by',
       'can',
       'couldn',
       "couldn't",
       'd',
       'did',
       'didn',
       "didn't",
       'do',
       'does',
       'doesn',
       "doesn't",
       'doing',
       'don',
       "don't",
       'down',
       'during',
       'each',
       'few',
       'for',
       'from',
       'further',
       'had',
       'hadn',
       "hadn't",
       'has',
       'hasn',
       "hasn't",
       'have',
       'haven',
       "haven't",
       'having',
       'he',
       "he'd",
       "he'll",
       'her',
       'here',
       'hers',
       'herself',
       "he's",
       'him',
       'himself',
       'his',
       'how',
       'i',
       "i'd",
       'if',
       "i'll",
       "i'm",
       'in',
       'into',
       'is',
       'isn',
       "isn't",
       'it',
       "it'd",
       "it'll",
       "it's",
       'its',
       'itself',
       "i've",
       'just',
       'll',
       'm',
       'ma',
       'me',
       'mightn',
       "mightn't",
       'more',
       'most',
       'mustn',
       "mustn't",
       'my',
       'myself',
       'needn',
       "needn't",
       'no',
       'nor',
       'not',
       'now',
       'o',
       'of',
       'off',
       'on',
       'once',
       'only',
       'or',
       'other',
       'our',
       'ours',
       'ourselves',
       'out',
       'over',
       'own',
       're',
       's',
       'same',
       'shan',
       "shan't",
       'she',
       "she'd",
       "she'll",
       "she's",
       'should',
       'shouldn',
       "shouldn't",
       "should've",
       'so',
       'some',
       'such',
       't',
       'than',
       'that',
       "that'll",
       'the',
       'their',
       'theirs',
       'them',
       'themselves',
       'then',
       'there',
       'these',
       'they',
       "they'd",
       "they'll",
       "they're",
       "they've",
       'this',
       'those',
       'through',
       'to',
       'too',
       'under',
       'until',
       'up',
       've',
       'very',
       'was',
       'wasn',
       "wasn't",
       'we',
       "we'd",
       "we'll",
       "we're",
       'were',
       'weren',
       "weren't",
       "we've",
       'what',
       'when',
       'where',
       'which',
       'while',
       'who',
       'whom',
       'why',
       'will',
       'with',
       'won',
       "won't",
       'wouldn',
       "wouldn't",
       'y',
       'you',
       "you'd",
       "you'll",
       'your',
       "you're",
       'yours',
       'yourself',
       'yourselves',
       "you've"]
```

[35]:
```
import string
string.punctuation
```

[35]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

[36]:
```
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
print(ps.stem("running"))
```

```
run
```

[37]:
```
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 stopwords.words('english') 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)
```

[38]:
```
transform_text("I'm gonna be home soon and i don't want to talk about this␣
  ↪stuff anymore tonight, k? I've cried enough today.")
```

[38]:
```
'gon na home soon want talk stuff anymor tonight k cri enough today'
```

[39]:
```
df['text'][0]
```

[39]:
```
'Go until jurong point, crazy.. Available only in bugis n great world la e
buffet… Cine there got amore wat…'
```

[40]:
```
df['transformed_text']=df['text'].apply(transform_text)
```

[41]:
```
df.head()
```

[41]:

```
   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

   num_words  num_sentences                                   transformed_text
0         24              2  go jurong point crazi avail bugi n great world…
1          8              2                            ok lar joke wif u oni
2         37              2  free entri 2 wkli comp win fa cup final tkt 21…
3         13              1                u dun say earli hor u c alreadi say
4         15              1             nah think goe usf live around though
```

[42]:

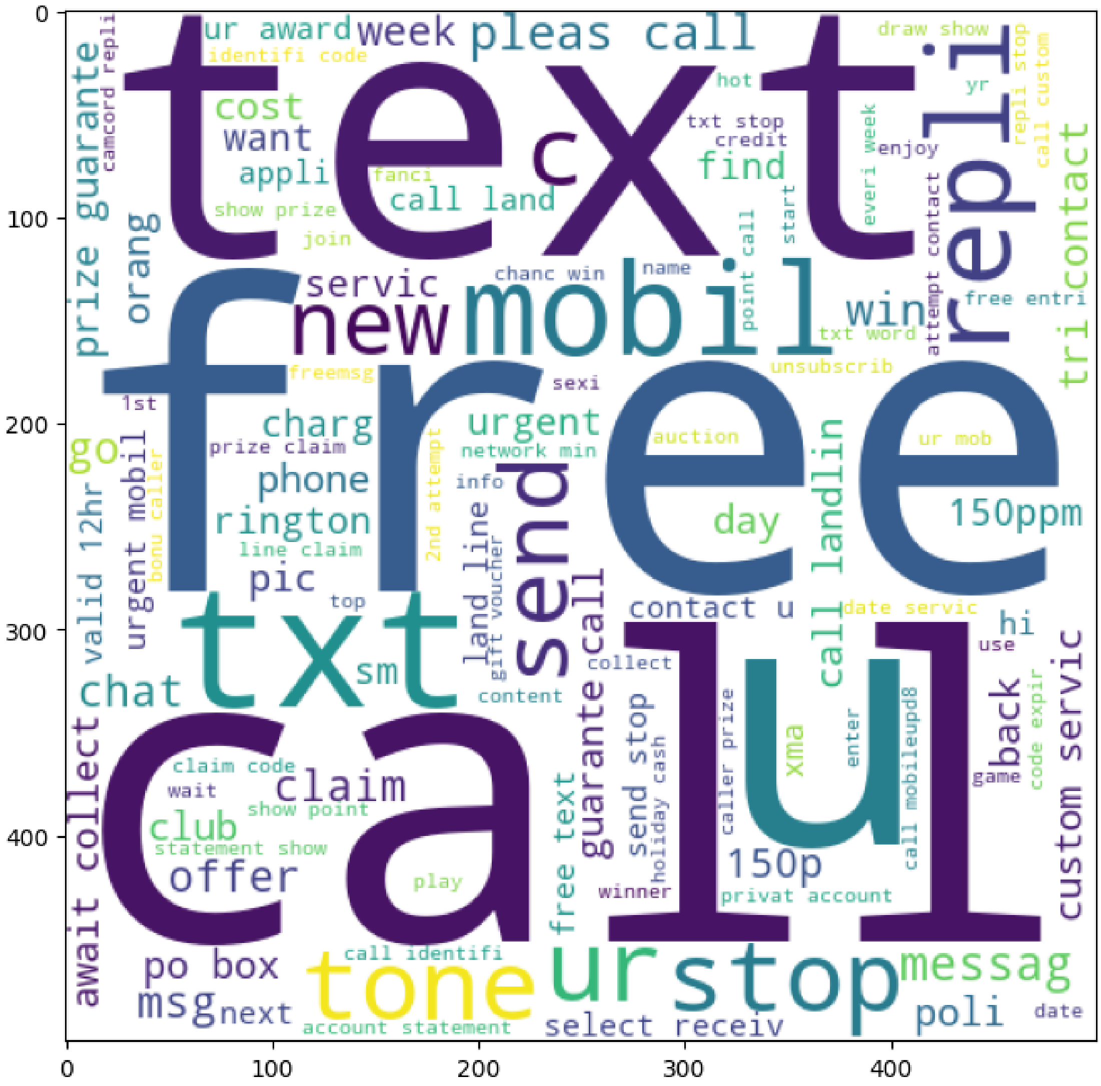
```python
from wordcloud import WordCloud
wc = WordCloud(width=500,height=500,min_font_size=10,background_color='white')
```

[43]:

```python
spam_wc=wc.generate(df[df['target']==1]['transformed_text'].str.cat(sep=" "))
plt.figure(figsize=(16,8))
plt.imshow(spam_wc)
```

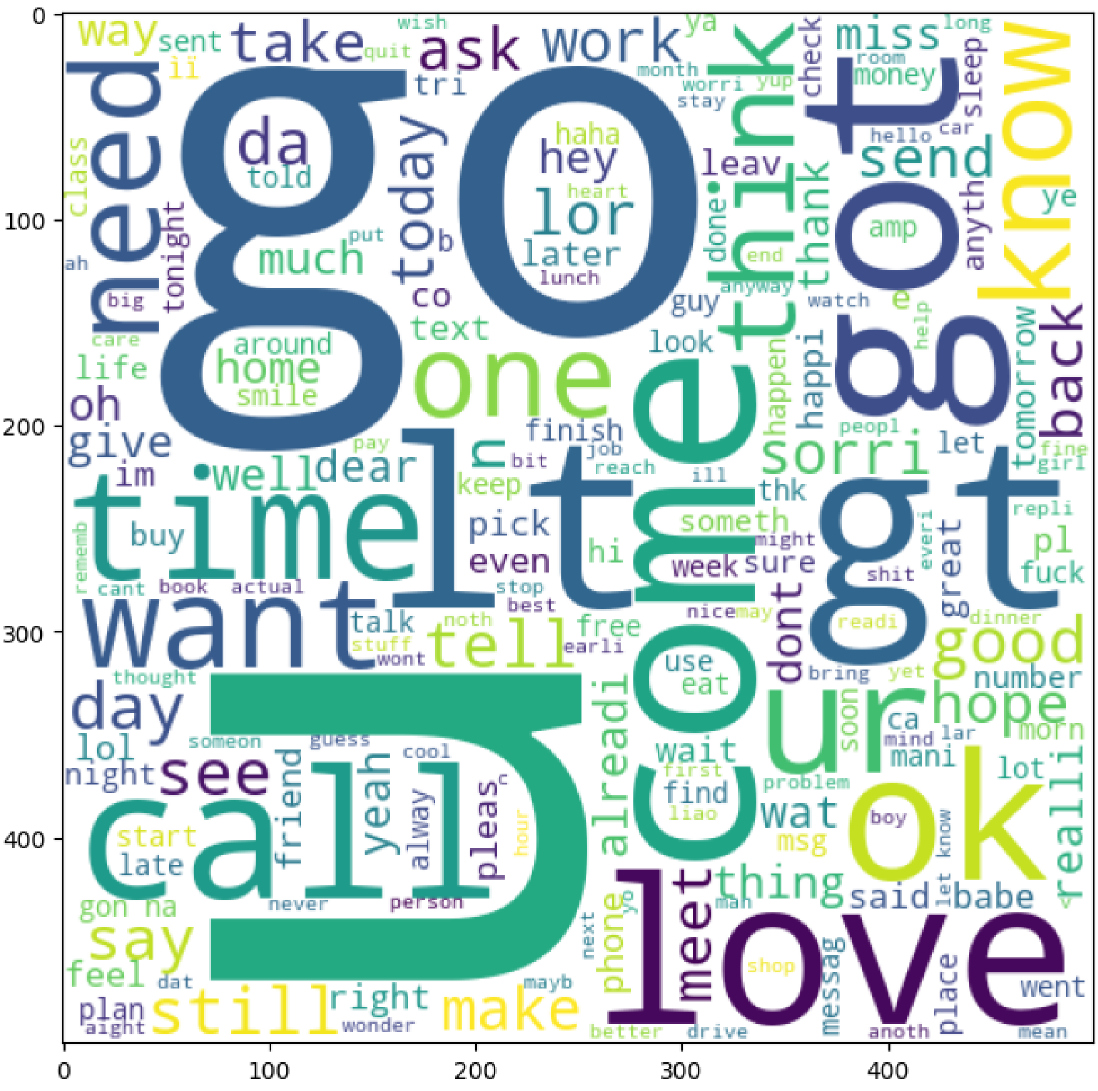
[43]:

```
<matplotlib.image.AxesImage at 0x110a74310>
```



```
[44]: ham_wc=wc.generate(df[df['target']==0]['transformed_text'].str.cat(sep=" "))
      plt.figure(figsize=(16,8))
      plt.imshow(ham_wc)
```

```
[44]: <matplotlib.image.AxesImage at 0x1100121d0>
```



[45]:

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

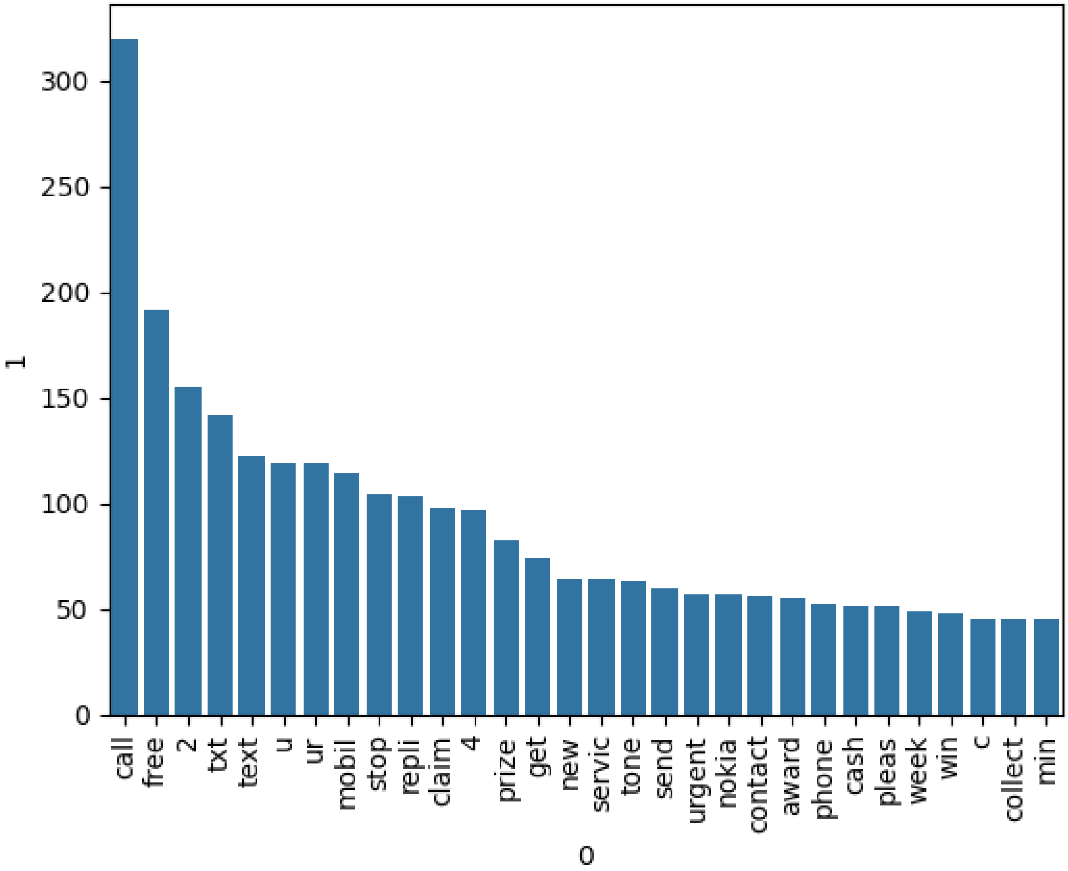
[46]:

```
len(spam_corpus)
```

[46]: 9939

[47]:

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



[48]:

```
ham_corpus=[]
for msg in df[df['target']==0]['transformed_text'].tolist():
  for word in msg.split():
    ham_corpus.append(word)
```

[49]:

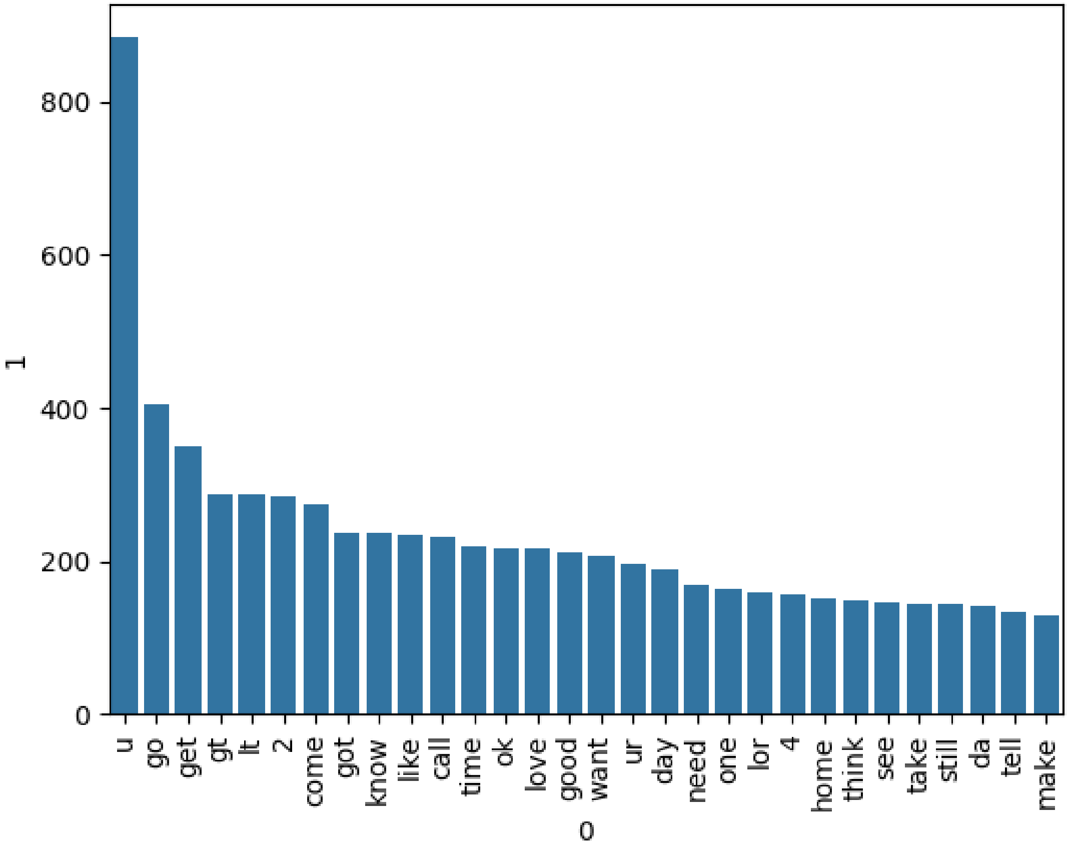
```
len(ham_corpus)
```

[49]:

```
35404
```

[50]:

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



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

```
[52]: X=tfidf.fit_transform(df['transformed_text']).toarray()
```

```
[53]: X.shape
```

```
[53]: (5169, 3000)
```

```
[54]: y = df['target'].values
```

```
[55]: from sklearn.model_selection import train_test_split
```

```
[56]: X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
        ↪2,random_state=2)
```

```
[57]: from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
      from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
```

[58]:

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

[59]:

```
gnb.fit(X_train,y_train)
y_pred1=gnb.predict(X_test)
print(accuracy_score(y_test,y_pred1))
print(confusion_matrix(y_test,y_pred1))
print(precision_score(y_test,y_pred1))
```

```
0.8694390715667312
[[788 108]
 [ 27 111]]
0.5068493150684932
```

[60]:

```
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))
```

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

[61]:

```
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))
```

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

[62]:

```
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
```

[63]:
```
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)
```

[64]:
```
clfs = {
    'SVC' : svc,
    'KN' : knc,
    'NB': mnb,
    'DT': dtc,
    'LR': lrc,
    'RF': rfc,
    'AdaBoost': abc,
    'BgC': bc,
    'ETC': etc,
    'GBDT':gbdt,
    'xgb':xgb
}
```

[65]:
```
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
```

[66]:
```
train_classifier(svc,X_train,y_train,X_test,y_test)
```

[66]:
```
(0.9758220502901354, 0.9747899159663865)
```

[67]:
```
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 -  0.9274661508704062
Precision -  0.8118811881188119
For  LR
Accuracy -  0.9584139264990329
Precision -  0.9702970297029703
For  RF
Accuracy -  0.9758220502901354
Precision -  0.9829059829059829
For  AdaBoost
Accuracy -  0.9245647969052224
Precision -  0.8488372093023255
For  BgC
Accuracy -  0.9584139264990329
Precision -  0.8682170542635659
For  ETC
Accuracy -  0.9748549323017408
Precision -  0.9745762711864406
For  GBDT
Accuracy -  0.9468085106382979
Precision -  0.9191919191919192
For  xgb
Accuracy -  0.9671179883945842
Precision -  0.9482758620689655
```

[68]:
```
performance_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy':
 ↪accuracy_scores,'Precision':precision_scores}).
 ↪sort_values('Precision',ascending=False)
```

[69]:
```
performance_df
```

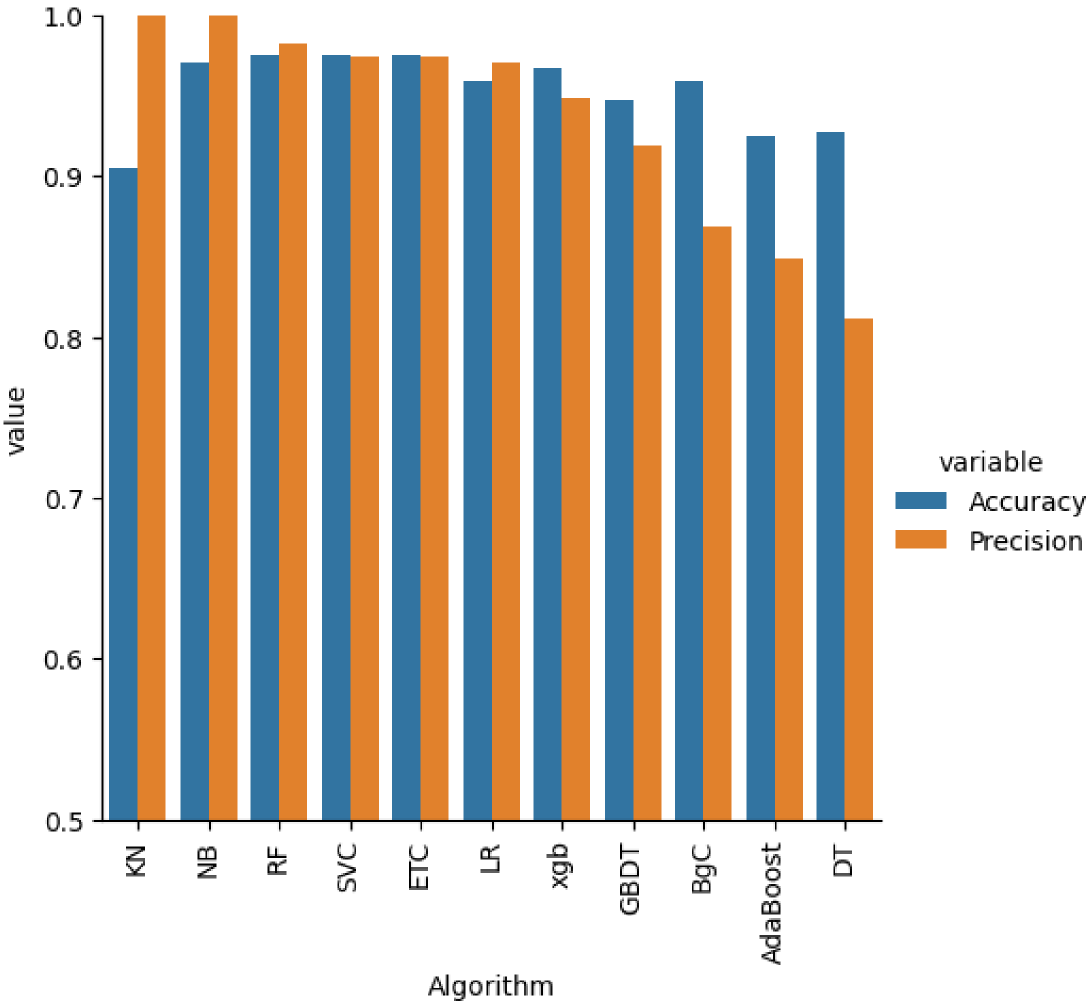
```
[69]:   Algorithm  Accuracy  Precision
     1         KN  0.905222   1.000000
     2         NB  0.970986   1.000000
     5         RF  0.975822   0.982906
     0        SVC  0.975822   0.974790
     8        ETC  0.974855   0.974576
     4         LR  0.958414   0.970297
     10       xgb  0.967118   0.948276
     9       GBDT  0.946809   0.919192
     7        BgC  0.958414   0.868217
     6   AdaBoost  0.924565   0.848837
     3         DT  0.927466   0.811881
```

```
[70]: performance_df1 = pd.melt(performance_df, id_vars = "Algorithm")
```

```
[71]: performance_df1
```

```
[71]:   Algorithm   variable     value
     0         KN   Accuracy  0.905222
     1         NB   Accuracy  0.970986
     2         RF   Accuracy  0.975822
     3        SVC   Accuracy  0.975822
     4        ETC   Accuracy  0.974855
     5         LR   Accuracy  0.958414
     6        xgb   Accuracy  0.967118
     7       GBDT   Accuracy  0.946809
     8        BgC   Accuracy  0.958414
     9   AdaBoost   Accuracy  0.924565
     10        DT   Accuracy  0.927466
     11        KN  Precision  1.000000
     12        NB  Precision  1.000000
     13        RF  Precision  0.982906
     14       SVC  Precision  0.974790
     15       ETC  Precision  0.974576
     16        LR  Precision  0.970297
     17       xgb  Precision  0.948276
     18      GBDT  Precision  0.919192
     19       BgC  Precision  0.868217
     20  AdaBoost  Precision  0.848837
     21        DT  Precision  0.811881
```

```
[72]: sns.catplot(x = 'Algorithm', y='value',
                     hue = 'variable',data=performance_df1, kind='bar',height=5)
      plt.ylim(0.5,1.0)
      plt.xticks(rotation='vertical')
      plt.show()
```



[73]:
```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_max_ft_3000':
↪accuracy_scores,'Precision_max_ft_3000':precision_scores}).
↪sort_values('Precision_max_ft_3000',ascending=False)
```

[74]:
```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_scaling':
↪accuracy_scores,'Precision_scaling':precision_scores}).
↪sort_values('Precision_scaling',ascending=False)
```

[75]:
```
new_df = performance_df.merge(temp_df,on='Algorithm')
```

[76]:
```
new_df_scaled = new_df.merge(temp_df,on='Algorithm')
```

[77]:
```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_num_chars':
↪accuracy_scores,'Precision_num_chars':precision_scores}).
↪sort_values('Precision_num_chars',ascending=False)
```

[78]:
```
new_df_scaled.merge(temp_df,on='Algorithm')
```

[78]:
```
   Algorithm  Accuracy  Precision  Accuracy_scaling_x  Precision_scaling_x  \
0         KN  0.905222   1.000000            0.905222             1.000000
1         NB  0.970986   1.000000            0.970986             1.000000
2         RF  0.975822   0.982906            0.975822             0.982906
3        SVC  0.975822   0.974790            0.975822             0.974790
4        ETC  0.974855   0.974576            0.974855             0.974576
5         LR  0.958414   0.970297            0.958414             0.970297
6        xgb  0.967118   0.948276            0.967118             0.948276
7       GBDT  0.946809   0.919192            0.946809             0.919192
8        BgC  0.958414   0.868217            0.958414             0.868217
9   AdaBoost  0.924565   0.848837            0.924565             0.848837
10        DT  0.927466   0.811881            0.927466             0.811881

    Accuracy_scaling_y  Precision_scaling_y  Accuracy_num_chars  \
0             0.905222             1.000000            0.905222
1             0.970986             1.000000            0.970986
2             0.975822             0.982906            0.975822
3             0.975822             0.974790            0.975822
4             0.974855             0.974576            0.974855
5             0.958414             0.970297            0.958414
6             0.967118             0.948276            0.967118
7             0.946809             0.919192            0.946809
8             0.958414             0.868217            0.958414
9             0.924565             0.848837            0.924565
10            0.927466             0.811881            0.927466

    Precision_num_chars
0              1.000000
1              1.000000
2              0.982906
3              0.974790
4              0.974576
5              0.970297
6              0.948276
7              0.919192
8              0.868217
9              0.848837
10             0.811881
```

[79]:
```
svc = SVC(kernel='sigmoid', gamma=1.0,probability=True)
mnb = MultinomialNB()
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)

from sklearn.ensemble import VotingClassifier
```

[80]:
```
voting = VotingClassifier(estimators=[('svm', svc), ('nb', mnb), ('et',␣
  ↪etc)],voting='soft')
```

[81]:
```
voting.fit(X_train,y_train)
```

[81]:
```
VotingClassifier(estimators=[('svm',
                              SVC(gamma=1.0, kernel='sigmoid',
                                  probability=True)),
                             ('nb', MultinomialNB()),
                             ('et',
                              ExtraTreesClassifier(n_estimators=50,
                                                   random_state=2))],
                 voting='soft')
```

[82]:
```
y_pred = voting.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

```
Accuracy 0.9816247582205029
Precision 0.9917355371900827
```

[83]:
```
estimators=[('svm', svc), ('nb', mnb), ('et', etc)]
final_estimator=RandomForestClassifier()
```

[84]:
```
from sklearn.ensemble import StackingClassifier
```

[85]:
```
clf = StackingClassifier(estimators=estimators, final_estimator=final_estimator)
```

[86]:
```
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 0.9806576402321083
Precision 0.946969696969697
```

[87]:
```
import pickle
pickle.dump(tfidf,open('vectorizer.pkl','wb'))
pickle.dump(mnb,open('model.pkl','wb'))
```