SMS-Spam-Detection

May 21, 2025

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

[3]:
```
df = pd.read_csv('spam.csv', encoding='ISO-8859-1')
```

[4]:
```
df.sample(5)
```

[4]:
```
        v1                                                 v2 Unnamed: 2  \
522    ham                         Were gonna go get some tacos        NaN
3000  spam  This message is free. Welcome to the new & imp…        NaN
1667   ham  So now my dad is gonna call after he gets out …        NaN
2676   ham  * Am on a train back from northampton so i'm a…        NaN
3683   ham                            Dad says hurry the hell up        NaN

     Unnamed: 3 Unnamed: 4
522         NaN        NaN
3000        NaN        NaN
1667        NaN        NaN
2676        NaN        NaN
3683        NaN        NaN
```

0.1 1. Data Cleaning

[5]:
```
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]:
```
# drop last three columns

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

[7]:
```
df.sample(5)
```

[7]:

	v1	v2
4964	ham	A few people are at the game, I'm at the mall …
953	ham	Also remember to get dobby's bowl from your car
5099	ham	Ah, well that confuses things, doesnt it? I th…
2600	ham	As usual..iam fine, happy & doing well..:)
2016	ham	Princess, is your kitty shaved or natural?

[8]:
```
# renaming the columns

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

[9]:
```
df.sample(5)
```

[9]:

	target	text
1598	ham	Daddy will take good care of you :)
796	spam	Orange customer, you may now claim your FREE C…
3324	ham	Nope… Juz off from work…
4924	ham	Ok… Let u noe when i leave my house.
3086	ham	So i asked how's anthony. Dad. And your bf

[10]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
```

[11]:
```
df['target']=encoder.fit_transform(df['target'])
```

[12]:
```
df.head()
```

[12]:

	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…

[13]:
```
# missing values

df.isnull().sum()
```

[13]:
```
target    0
text      0
dtype: int64
```

[14]:
```
# check for dublicate values

df.duplicated().sum()
```

[14]:
```
403
```

[15]:
```
# remove duplicates

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

[16]:
```
df.duplicated().sum()
```

[16]:
```
0
```

[17]:
```
df.shape
```

[17]:
```
(5169, 2)
```

0.2 EDA

[18]:
```
df.head()
```

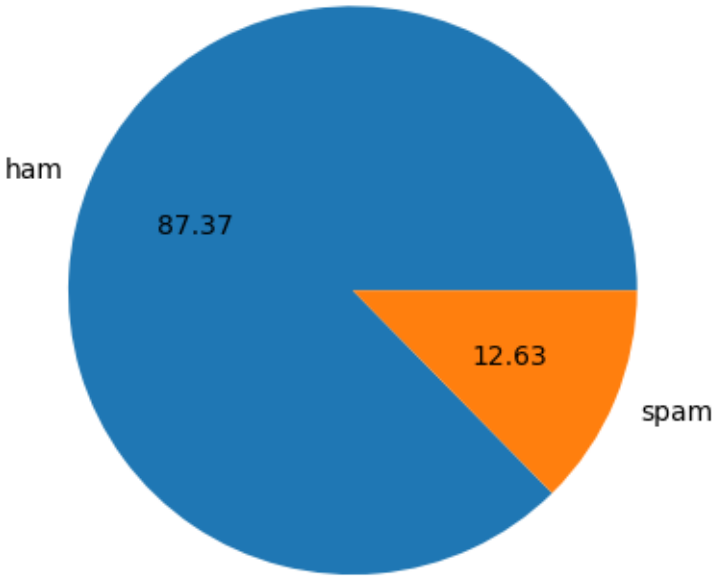
[18]:

	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…

[19]:
```
df['target'].value_counts()
```

[19]:
```
target
0    4516
1     653
Name: count, dtype: int64
```

[20]:
```
import matplotlib.pyplot as plt
plt.pie(df['target'].value_counts(), labels=['ham','spam'], autopct = '%0.2f')
plt.show()
```



Data is imbalanced

[22]:
```
import nltk
```

[23]:
```
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to C:\Users\JASHANDEEP
[nltk_data]     SINGH/nltk_data…
[nltk_data]   Package punkt_tab is already up-to-date!
```

[23]:
```
True
```

[24]:
```
# number of characters

df['num_cahracters'] = df['text'].apply(len)
df.head()
```

[24]:

	target	text	num_cahracters
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

[25]:
```
# number of words

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

[25]:
```
   target                                                text  num_cahracters  \
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
```

[26]:
```
# number of sentences

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

[26]:
```
   target                                                text  num_cahracters  \
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
```

[27]:
```
df[['num_cahracters','num_words','num_sentences']].describe()
```

[27]:
```
       num_cahracters    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
```

[28]:
```
# ham
df[df['target']==0][['num_cahracters','num_words','num_sentences']].describe()
```

[28]:
```
       num_cahracters    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
```

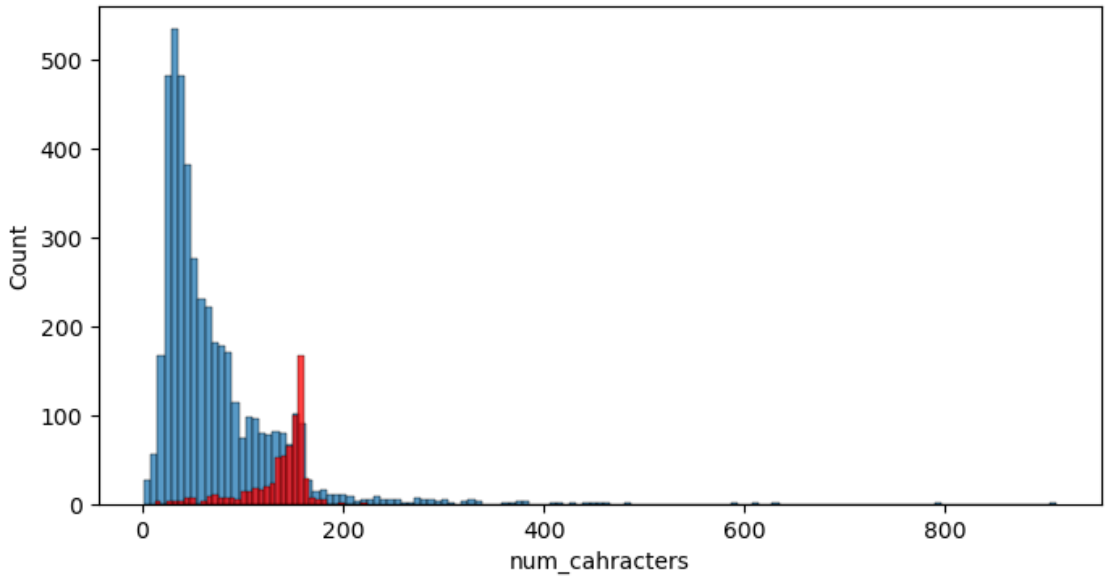
[29]:
```
# spam
df[df['target']==1][['num_cahracters','num_words','num_sentences']].describe()
```

[29]:
```
       num_cahracters   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
```

[30]:
```
import seaborn as sns
```

[31]:
```
plt.figure(figsize = (8,4))
sns.histplot(df[df['target']==0]['num_cahracters'])
sns.histplot(df[df['target']==1]['num_cahracters'], color='red')
```

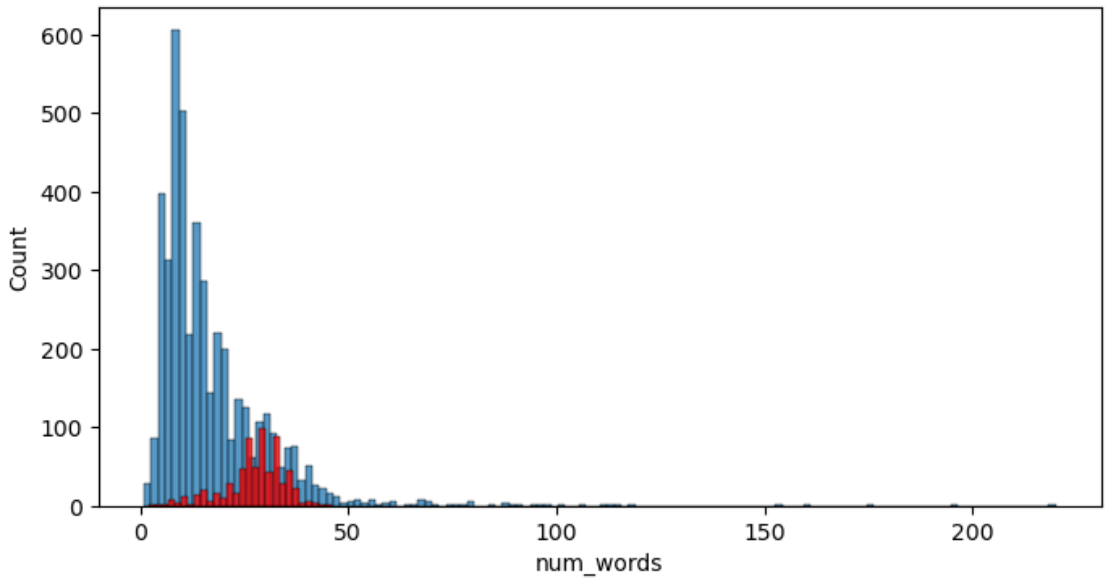
[31]:
```
<Axes: xlabel='num_cahracters', ylabel='Count'>
```



Spam messages are generally consiting more characters than the ham messages

[32]:
```
plt.figure(figsize = (8,4))
sns.histplot(df[df['target']==0]['num_words'])
sns.histplot(df[df['target']==1]['num_words'], color='red')
```

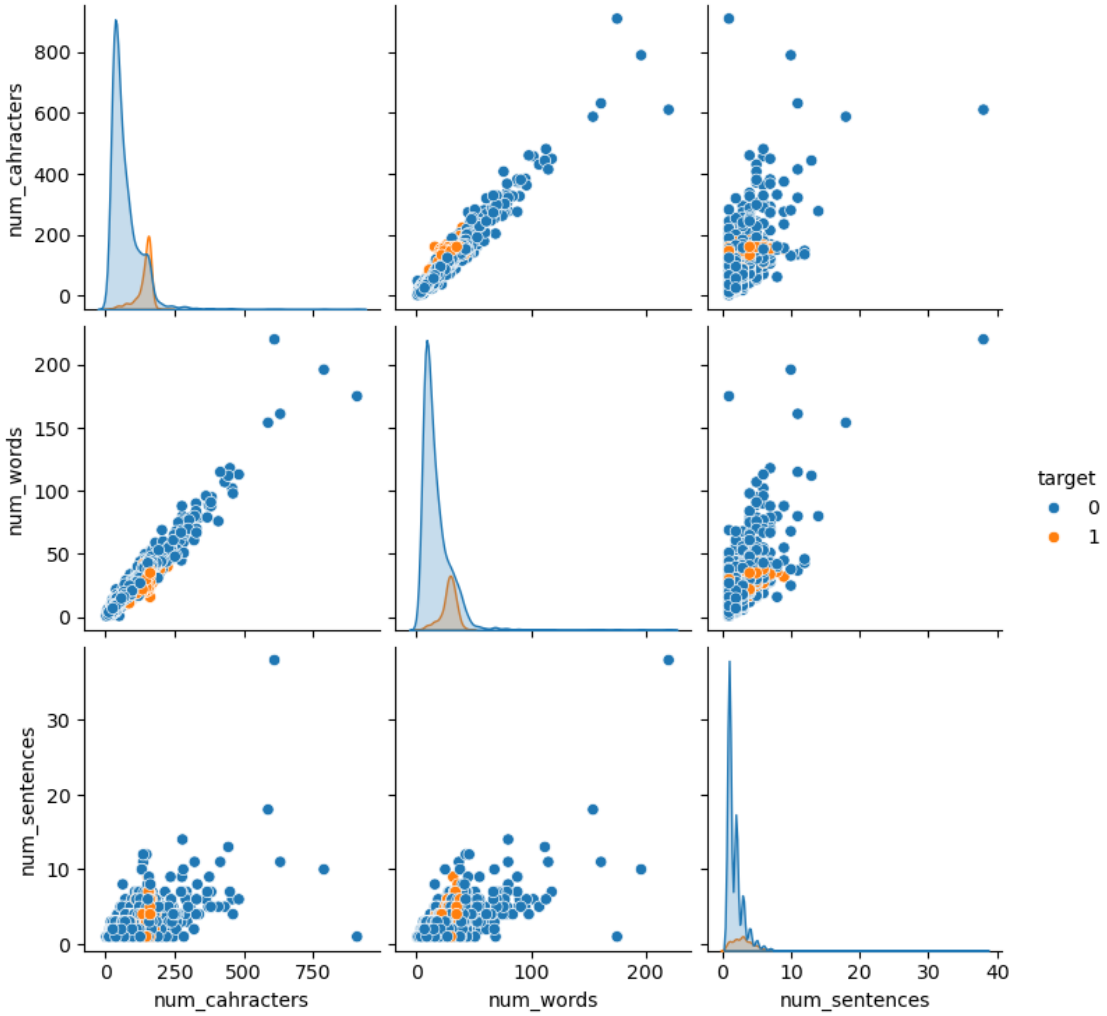
[32]: <Axes: xlabel='num_words', ylabel='Count'>



Spam messages are generally consiting more words than the ham messages

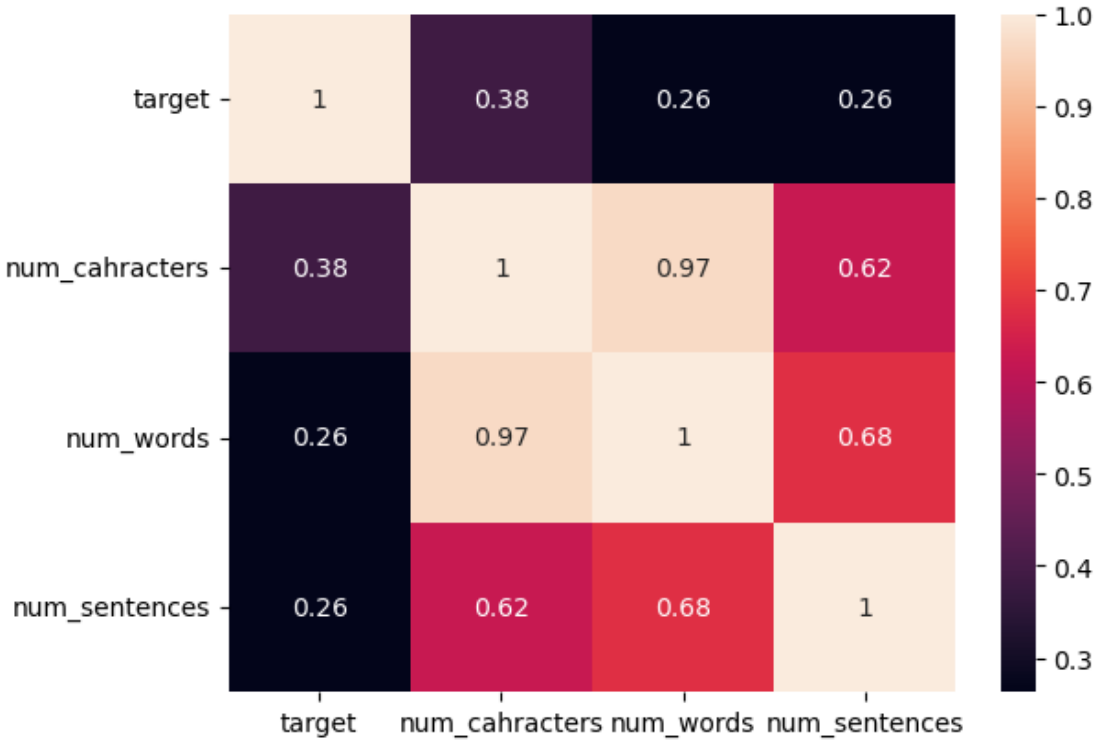
[33]:
```
sns.pairplot(df,hue='target')
```

[33]:
```
<seaborn.axisgrid.PairGrid at 0x21d98916fc0>
```



[34]:
```
sns.heatmap(df.select_dtypes(include=['number']).corr(),annot=True)
```

[34]:
```
<Axes: >
```



The target is showing higher positive correlation with no. of words and no. of charachters which means its highly dependent on these two factors to determine weather a message is a spam or ham.

0.3 3. Data Preprocessing

1. Lower Case
2. Tokenization
3. Removing Special Characters
4. Removing stop words and punctuation
5. Stemming

[35]:
```
from nltk.corpus import stopwords
import string
from nltk.stem.porter import PorterStemmer
p = PorterStemmer()
```

[36]:
```
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(p.stem(i))

    return " ".join(y)
```

[37]:
```
df['transformed_text'] = df['text'].apply(transform_text)
df.head()
```

[37]:

	target	text	num_cahracters	\
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	

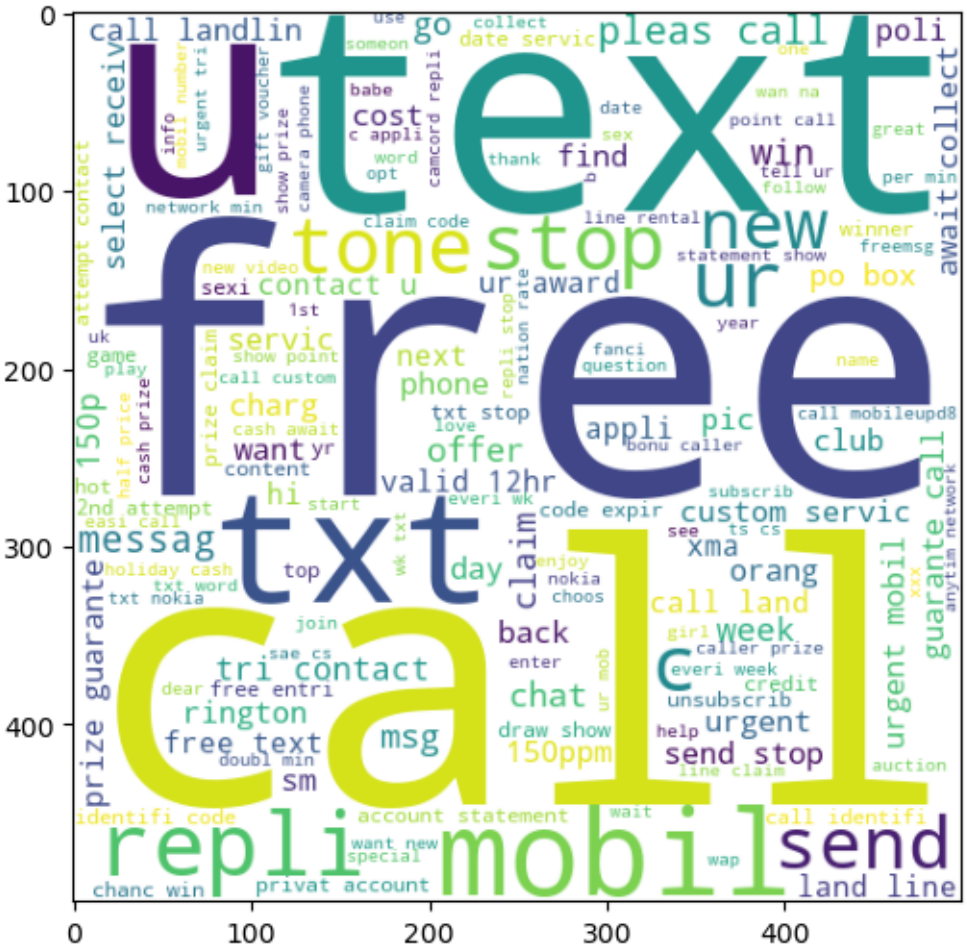
	num_words	num_sentences	transformed_text
0	24	2	go jurong point crazi avail bugi n great world…
1	8	2	ok lar joke wif u oni
2	37	2	free entri 2 wkli comp win fa cup final tkt 21…
3	13	1	u dun say earli hor u c alreadi say
4	15	1	nah think goe usf live around though

[113]:
```
from wordcloud import WordCloud
wc = WordCloud(width=500,height=500,min_font_size=10,background_color='white')
```

[39]:
```
# word cloud for spam messages
spam_wc = wc.generate(df[df['target']==1]['transformed_text'].str.cat(sep=" "))
```

[40]:
```
plt.figure(figsize=(15,6))
plt.imshow(spam_wc)
```

[40]:
```
<matplotlib.image.AxesImage at 0x21d9aa09f10>
```



[41]:
```
# word cloud for ham messages
ham_wc = wc.generate(df[df['target']==0]['transformed_text'].str.cat(sep=" "))
plt.figure(figsize=(15,6))
plt.imshow(ham_wc)
```

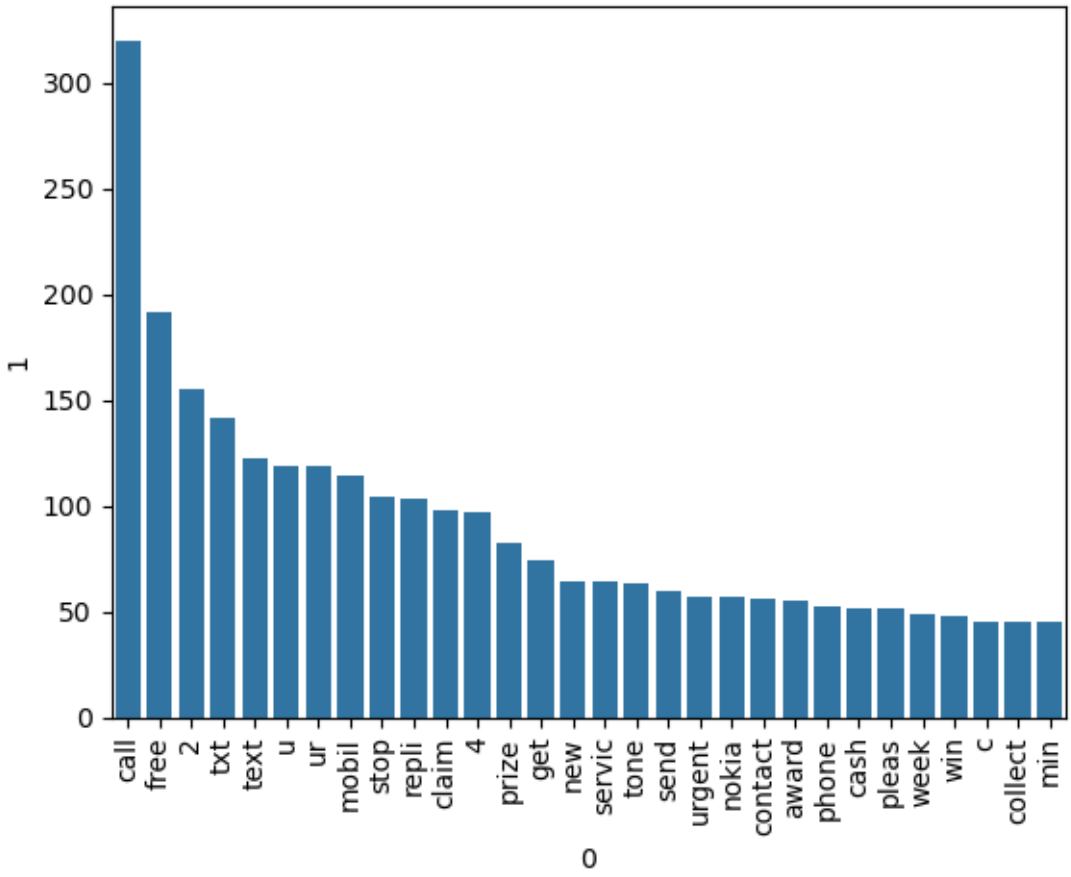
[41]: <matplotlib.image.AxesImage at 0x21d9bf21220>

[42]:
```
# Top 30 words
spam_corpus = []
for msg in df[df['target']==1]['transformed_text'].tolist():
    for word in msg.split():
        spam_corpus.append(word)

len(spam_corpus)
```

[42]: 9939

[43]:
```
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()
```

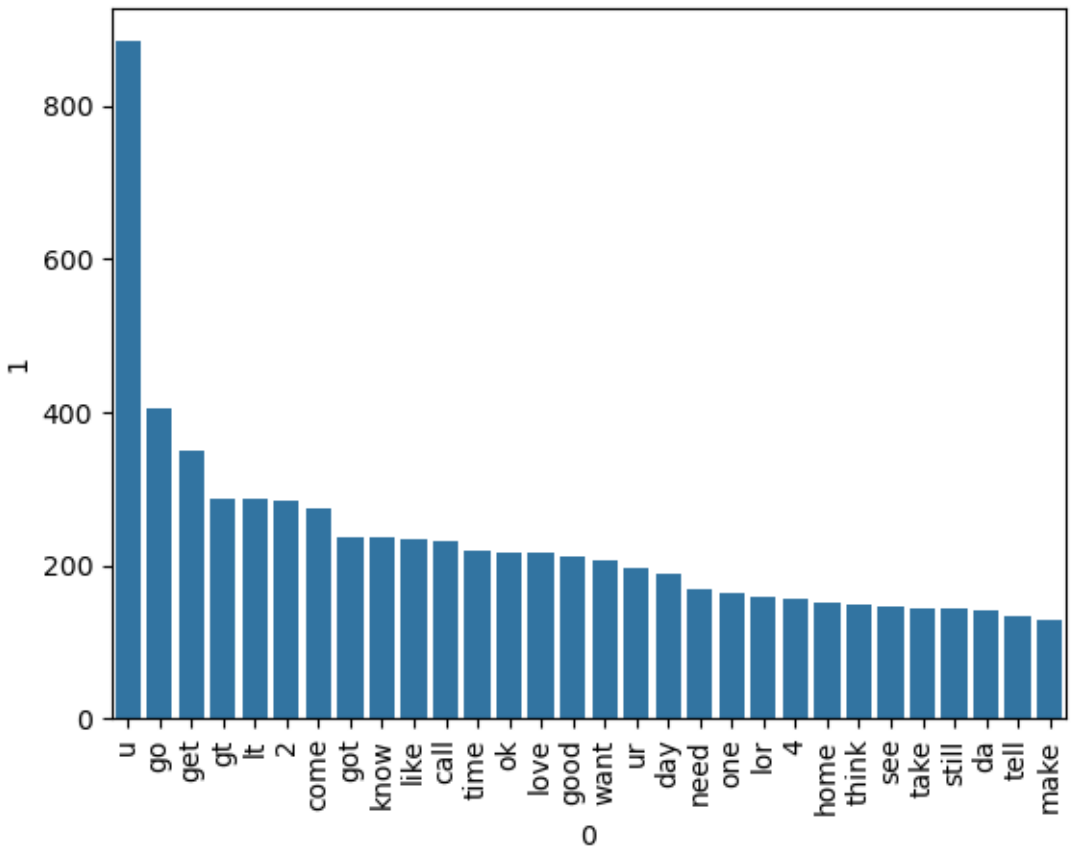


[44]:
```
ham_corpus = []
for msg in df[df['target'] == 0]['transformed_text'].tolist():
    for word in msg.split():
        ham_corpus.append(word)
```

[45]:
```
len(ham_corpus)
```

[45]: 35404

[46]:
```
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()
```



0.4 4. Model Building

[101]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
cv = CountVectorizer()
tfidf = TfidfVectorizer(max_features=3000)
```

[102]:
```
x = tfidf.fit_transform(df['transformed_text']).toarray()
```

[103]:
```
x.shape
```

[103]:
```
(5169, 3000)
```

[50]:
```
#from sklearn.preprocessing import MinMaxScaler
#scaler = MinMaxScaler()
#x = scaler.fit_transform(x)
```

[51]:
```
#appending the num_character col to x
#x = np.hstack((x,df['num_cahracters'].values.reshape(-1,1)))
```

[104]:
```
y = df['target'].values
```

[105]:
```
y
```

[105]:
```
array([0, 0, 1, …, 0, 0, 0])
```

[106]:
```
from sklearn.model_selection import train_test_split
```

[107]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,␣
↪random_state = 2)
```

[108]:
```
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
```

[109]:
```
gnb = GaussianNB()
mnb = MultinomialNB()
bnb = BernoulliNB()
```

[81]:
```
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
```

[111]:
```
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
```

[83]:
```
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
```

[84]:
```
# tfidf - mnb
```

[86]:
```
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
```

[87]:
```
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)
```

[88]:
```
clfs = {
    'SVC' : svc,
    'KN' : knc,
    'NB': mnb,
    'DT': dtc,
    'LR': lrc,
    'RF': rfc,
    'AdaBoost': abc,
    'BgC': bc,
    'ETC': etc,
    'GBDT':gbdt,
    'xgb':xgb
}
```

[89]:
```
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
```

[90]:
```
train_classifier(svc,x_train,y_train,x_test,y_test)
```

[90]:
```
(0.9758220502901354, 0.9747899159663865)
```

[91]:
```
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 -  0.9294003868471954
Precision -  0.8282828282828283
For  LR
Accuracy -  0.9584139264990329
Precision -  0.9702970297029703
For  RF
Accuracy -  0.9758220502901354
Precision -  0.9829059829059829
```

```
C:\Users\JASHANDEEP SINGH\Anaconda\Lib\site-
packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
```

```
For  AdaBoost
Accuracy -  0.960348162475822
```

```
Precision -  0.9292035398230089
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	0.900387	1.000000
2	NB	0.959381	1.000000
5	RF	0.971954	1.000000
8	ETC	0.972921	0.982456
0	SVC	0.972921	0.974138
10	xgb	0.974855	0.951613
6	AdaBoost	0.961315	0.945455
4	LR	0.951644	0.940000
9	GBDT	0.952611	0.923810
3	DT	0.938104	0.862745
7	BgC	0.958414	0.862595

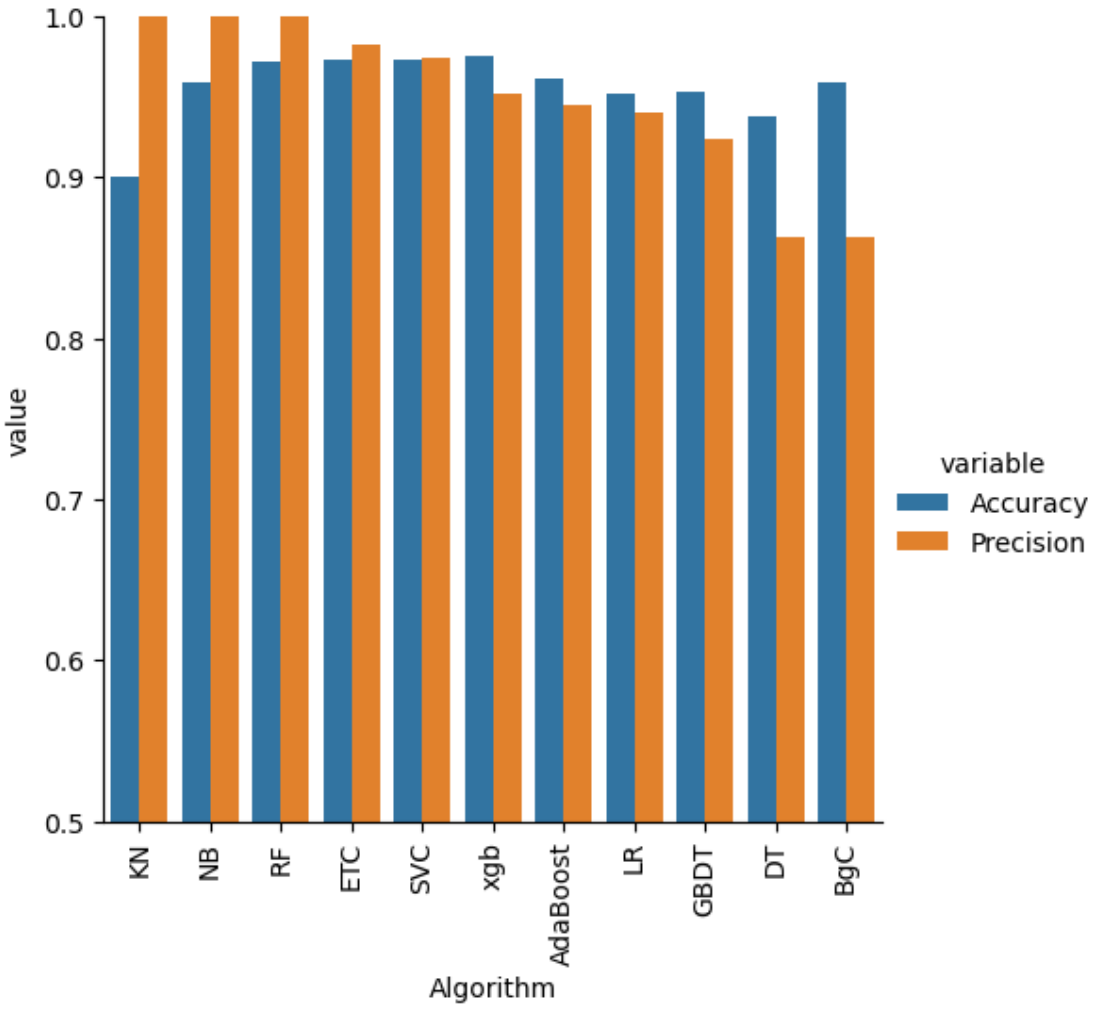
[70]:
```
performance_df1 = pd.melt(performance_df, id_vars = "Algorithm")
performance_df1
```

[70]:

	Algorithm	variable	value
0	KN	Accuracy	0.900387
1	NB	Accuracy	0.959381
2	RF	Accuracy	0.971954
3	ETC	Accuracy	0.972921
4	SVC	Accuracy	0.972921
5	xgb	Accuracy	0.974855
6	AdaBoost	Accuracy	0.961315
7	LR	Accuracy	0.951644
8	GBDT	Accuracy	0.952611
9	DT	Accuracy	0.938104
10	BgC	Accuracy	0.958414

```
11         KN  Precision  1.000000
12         NB  Precision  1.000000
13         RF  Precision  1.000000
14        ETC  Precision  0.982456
15        SVC  Precision  0.974138
16        xgb  Precision  0.951613
17   AdaBoost  Precision  0.945455
18         LR  Precision  0.940000
19       GBDT  Precision  0.923810
20         DT  Precision  0.862745
21        BgC  Precision  0.862595
```

[71]:
```
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()
```



[ ]:
```
# model improve
# 1. Change the max_features parameter of TfIdf
```

[92]:
```
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)
```

[93]:
```
new_df = performance_df.merge(temp_df,on='Algorithm')
```

[94]:
```
new_df
```

[94]:

	Algorithm	Accuracy	Precision	Accuracy_max_ft_3000	Precision_max_ft_3000
0	KN	0.900387	1.000000	0.905222	1.000000
1	NB	0.959381	1.000000	0.970986	1.000000
2	RF	0.971954	1.000000	0.975822	0.982906
3	ETC	0.972921	0.982456	0.974855	0.974576
4	SVC	0.972921	0.974138	0.975822	0.974790
5	xgb	0.974855	0.951613	0.967118	0.948276
6	AdaBoost	0.961315	0.945455	0.960348	0.929204
7	LR	0.951644	0.940000	0.958414	0.970297
8	GBDT	0.952611	0.923810	0.946809	0.919192
9	DT	0.938104	0.862745	0.929400	0.828283
10	BgC	0.958414	0.862595	0.958414	0.868217

Initially, models like Multinomial Naive Bayes (MNB), K-Nearest Neighbors (KNN), and Random Forest (RF) achieved perfect precision (1.000), but with slightly lower accuracy — around 95% to 97%. To improve performance, we set the max_features parameter in the TfidfVectorizer to 3000, which boosted the accuracy of the MNB model from 95.9% to 97.1%, while maintaining a perfect precision score.

Since the dataset is imbalanced (with far more ham messages than spam), it is crucial to prioritize precision. A high precision ensures that when the model predicts a message as spam, it is almost always correct — minimizing false positives. Considering its high precision, improved accuracy, and simplicity, Multinomial Naive Bayes (MNB) with max_features=3000 is selected as the final model for deployment.

[112]:
```
import pickle
pickle.dump(tfidf,open('vectorizer.pkl','wb'))
pickle.dump(mnb,open('model.pkl','wb'))
```