SMS Spam classifier

December 18, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[2]:
```
df = pd.read_csv('sms.csv')
```

[3]:
```
df.head()
```

[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]:
```
df.columns
```

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

[5]:
```
df.shape
```

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

1 1. Data Cleaning

[6]:
```
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
```

[7]:
```
# dropping last 3 columns
df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],axis=1,inplace=True)
```

[8]:
```
df.head()
```

[8]:

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

[9]:
```
# renaming columns
df.rename(columns={'v1':'target','v2':'message'},inplace=True)
```

[10]:
```
df.head()
```

[10]:

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

[11]:
```
df.isnull().sum()
```

[11]:
```
target     0
message    0
dtype: int64
```

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

[12]:
```
403
```

[13]:
```
# removing duplicate values
df = df.drop_duplicates(keep = 'first')
```

[14]:
```
df.duplicated().sum()
```

[14]:
```
0
```

[15]:
```
df.shape
```

[15]:
```
(5169, 2)
```

[16]:
```
from sklearn.preprocessing import LabelEncoder
```

[17]:
```
encoder = LabelEncoder()
```

[18]:
```
df['target'] = encoder.fit_transform(df['target'])
```

[19]:
```
df.head()
```

[19]:

	target	message
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...

2 2. Exploratory Data Analysis

[20]:
```
df.head()
```

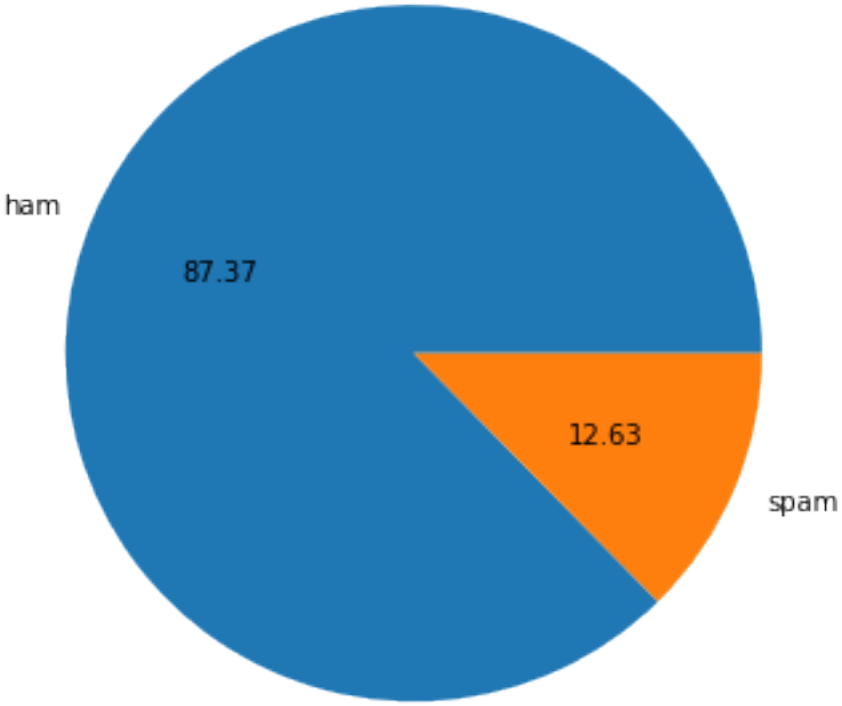
[20]:

	target	message
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...

[21]:
```
df['target'].value_counts()
```

[21]:
```
0    4516
1     653
Name: target, dtype: int64
```

[22]:
```
plt.figure(figsize=(10,6))
plt.pie(df['target'].value_counts(), labels=['ham','spam'], autopct='%0.2f')
plt.show()
```



[23]:
```
import nltk
```

[24]:
```
df['num_characters'] = df['message'].apply(len)
```

[25]:
```
df.head()
```

[25]:

	target	message	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

[26]:
```
df['num_words'] = df['message'].apply(lambda x: len(nltk.word_tokenize(x)))
```

[27]:
```
df['num_sentences'] = df['message'].apply(lambda x: len(nltk.sent_tokenize(x)))
```

[28]:
```
df.columns
```

[28]:
```
Index(['target', 'message', 'num_characters', 'num_words', 'num_sentences'],
dtype='object')
```

[29]:
```
df[['num_characters', 'num_words', 'num_sentences']].describe()
```

[29]:

	num_characters	num_words	num_sentences
count	5169.000000	5169.000000	5169.000000
mean	78.923776	18.456375	1.962275
std	58.174846	13.323322	1.433892
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

[30]:
```
#ham
df[df['target']==0].describe()
```

[30]:

	target	num_characters	num_words	num_sentences
count	4516.0	4516.000000	4516.000000	4516.000000
mean	0.0	70.456820	17.123339	1.815545
std	0.0	56.356802	13.491315	1.364098
min	0.0	2.000000	1.000000	1.000000
25%	0.0	34.000000	8.000000	1.000000
50%	0.0	52.000000	13.000000	1.000000
75%	0.0	90.000000	22.000000	2.000000
max	0.0	910.000000	220.000000	38.000000

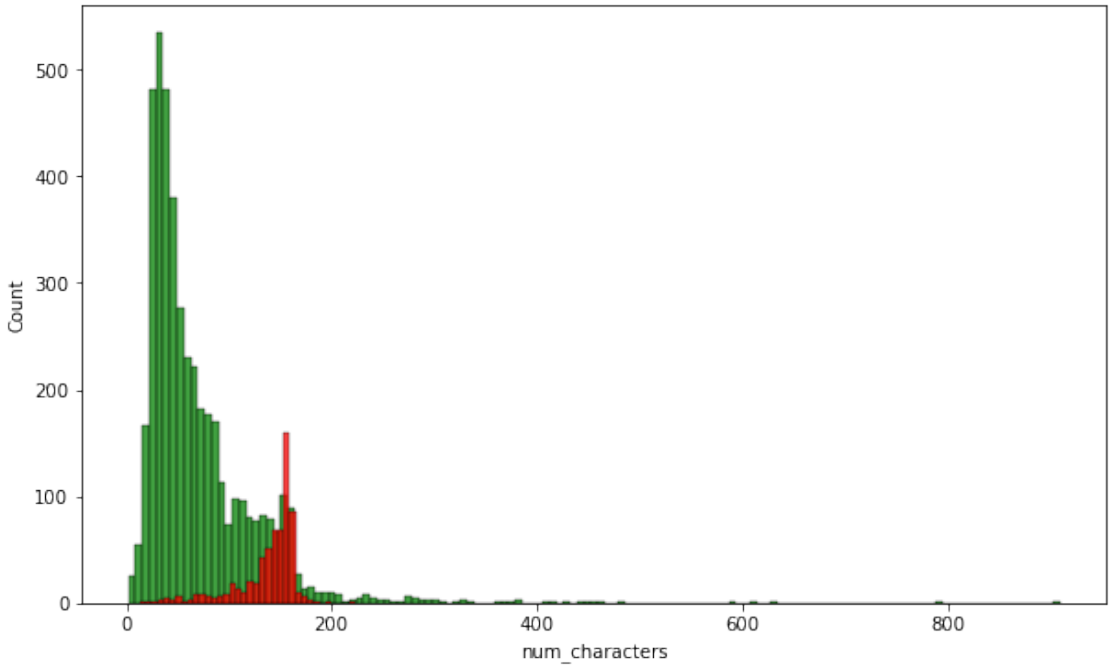
[31]:
```
#spam
df[df['target']==1].describe()
```

[31]:

	target	num_characters	num_words	num_sentences
count	653.0	653.000000	653.000000	653.000000
mean	1.0	137.479326	27.675345	2.977029
std	0.0	30.014336	7.011513	1.493676
min	1.0	13.000000	2.000000	1.000000
25%	1.0	131.000000	25.000000	2.000000
50%	1.0	148.000000	29.000000	3.000000
75%	1.0	157.000000	32.000000	4.000000
max	1.0	223.000000	46.000000	9.000000

[32]:
```
plt.figure(figsize=(10,6))
sns.histplot(df[df['target']==0]['num_characters'],color='g')
sns.histplot(df[df['target']==1]['num_characters'],color='r')
```

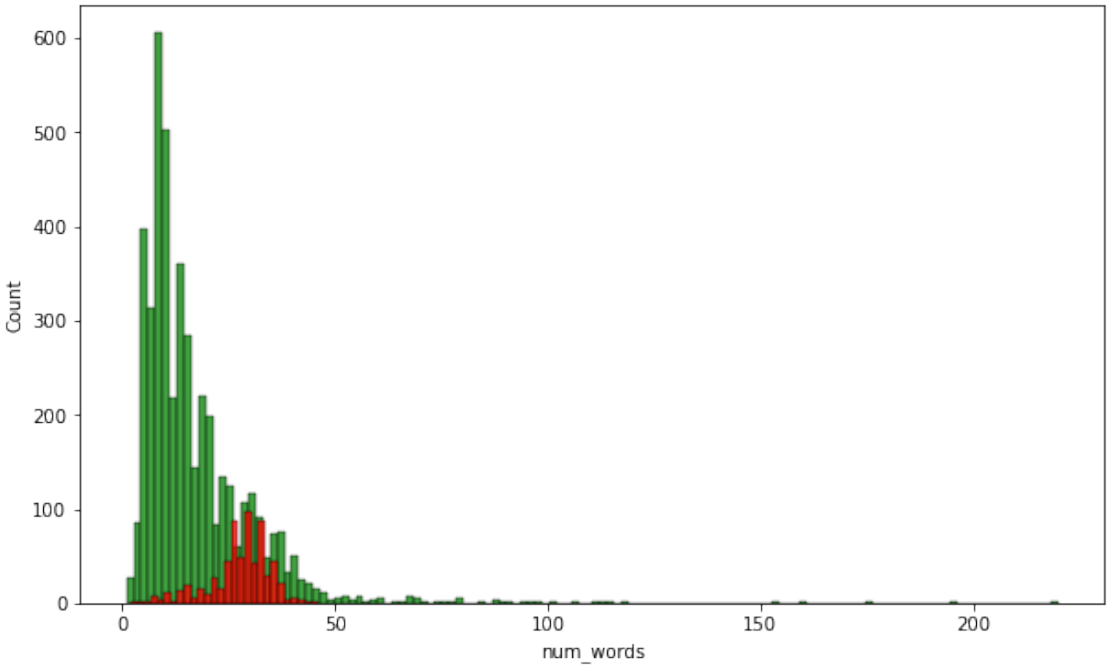
[32]:
```
<AxesSubplot:xlabel='num_characters', ylabel='Count'>
```



we can see that mean length of spam messages is greater than ham messages.With this We can confirm that usually spam messages are bigger than ham messages.

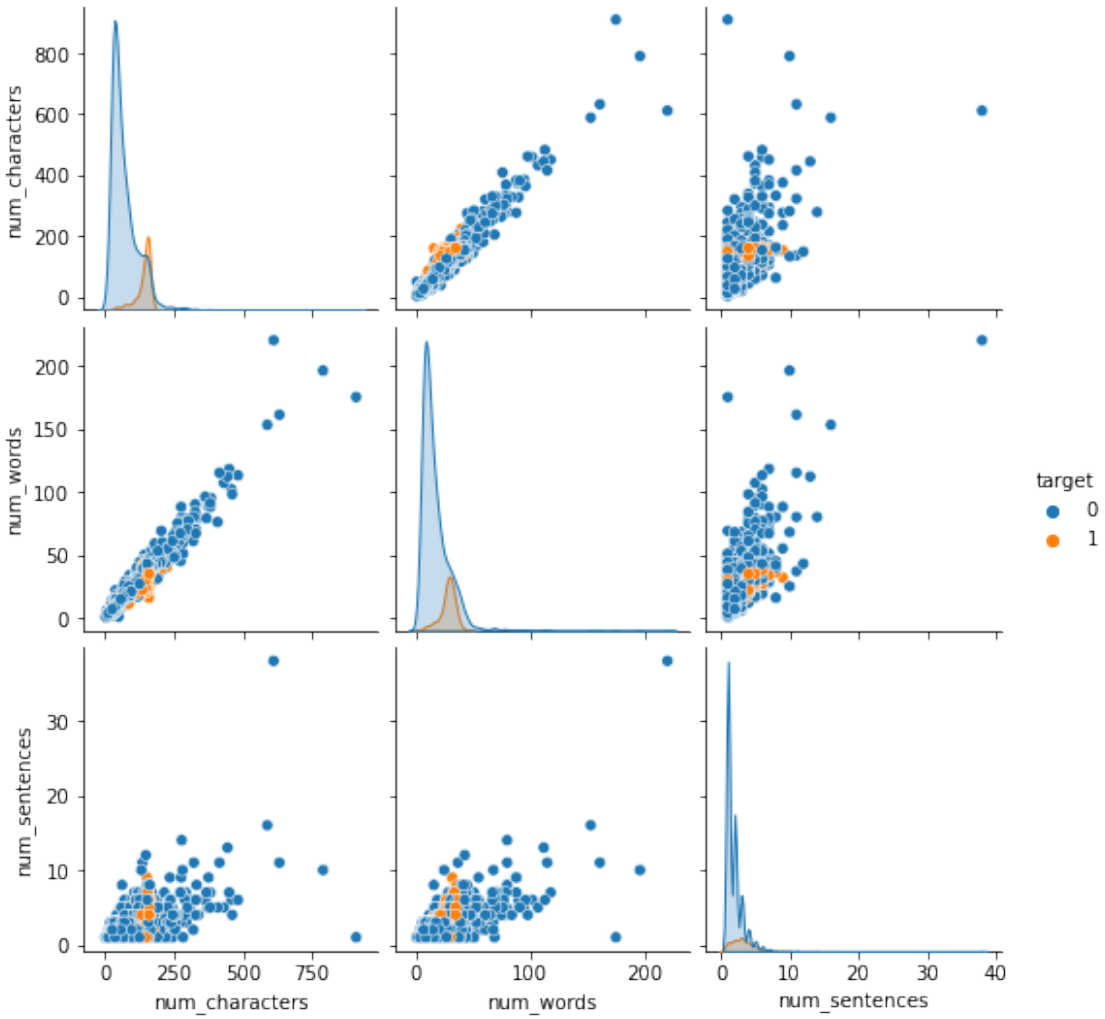
[33]:
```
plt.figure(figsize=(10,6))
sns.histplot(df[df['target']==0]['num_words'],color='g')
sns.histplot(df[df['target']==1]['num_words'],color='r')
```

[33]:
```
<AxesSubplot:xlabel='num_words', ylabel='Count'>
```



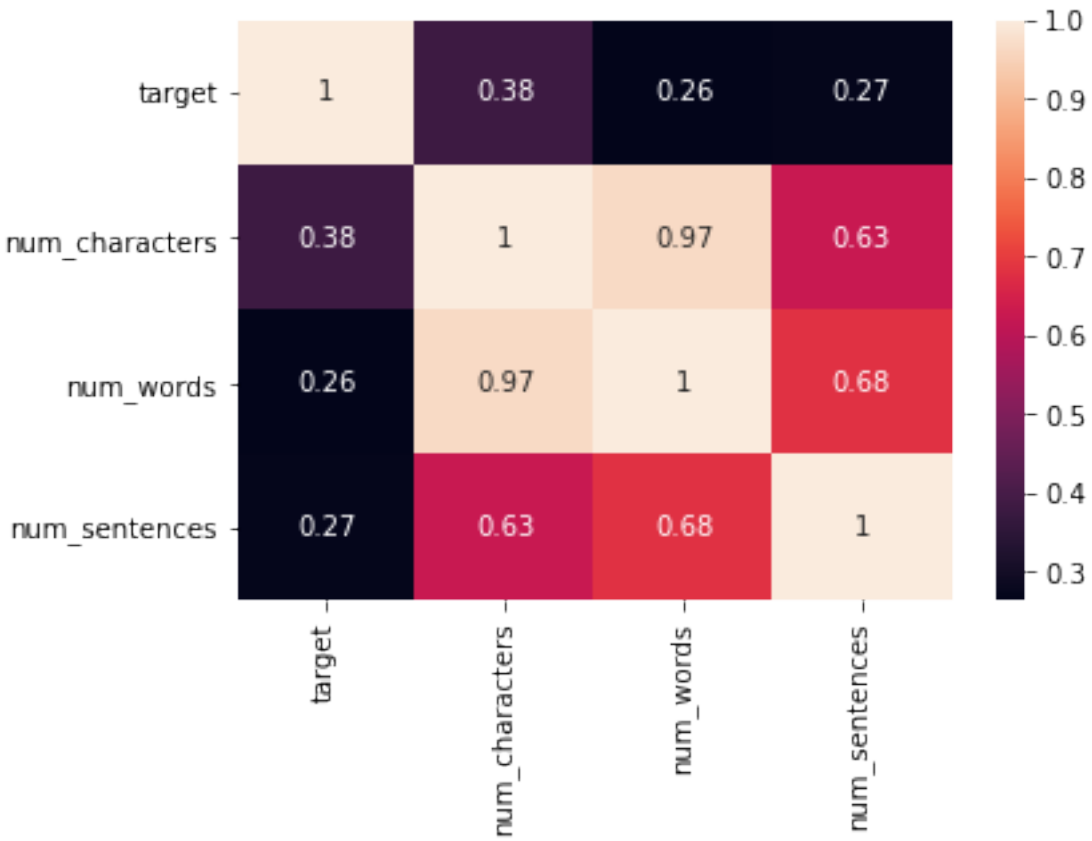
[34]:
```
sns.pairplot(df,hue='target')
```

[34]: <seaborn.axisgrid.PairGrid at 0x1e8c1df9cd0>



[35]:
```
sns.heatmap(df.corr(),annot=True)
```

[35]: <AxesSubplot:>



3 3. Text Preprocessing

- Lower case
- Tokenization
- Removing Special characters
- Removing stop words and punctuation
- stemming

[36]:
```
from nltk.corpus import stopwords
```

[37]:
```
from nltk.stem.porter import PorterStemmer
```

[38]:
```
stemmer = PorterStemmer()
```

[39]:
```
import string
string.punctuation
```

[39]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

[40]:
```
def text_preprocessing(text):
    text = text.lower()
    text = nltk.word_tokenize(text)
    a = []
    for i in text:
        if i.isalnum():
            a.append(i)

    text = a[:]
    a.clear()

    for i in text:
        if i not in stopwords.words('english') and i not in string.punctuation:
            a.append(i)

    text = a[:]
    a.clear()

    for i in text:
        a.append(stemmer.stem(i))

    return ' '.join(a)
```

[41]:
```
text_preprocessing('Okay name ur price as long as its legal! Wen can I pick them␣
↪up? Y u ave x ams xx')
```

[41]:
```
'okay name ur price long legal wen pick u ave x am xx'
```

[42]:
```
df['transformed_text'] = df['message'].apply(text_preprocessing)
```

[43]:
```
df.head()
```

[43]:

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

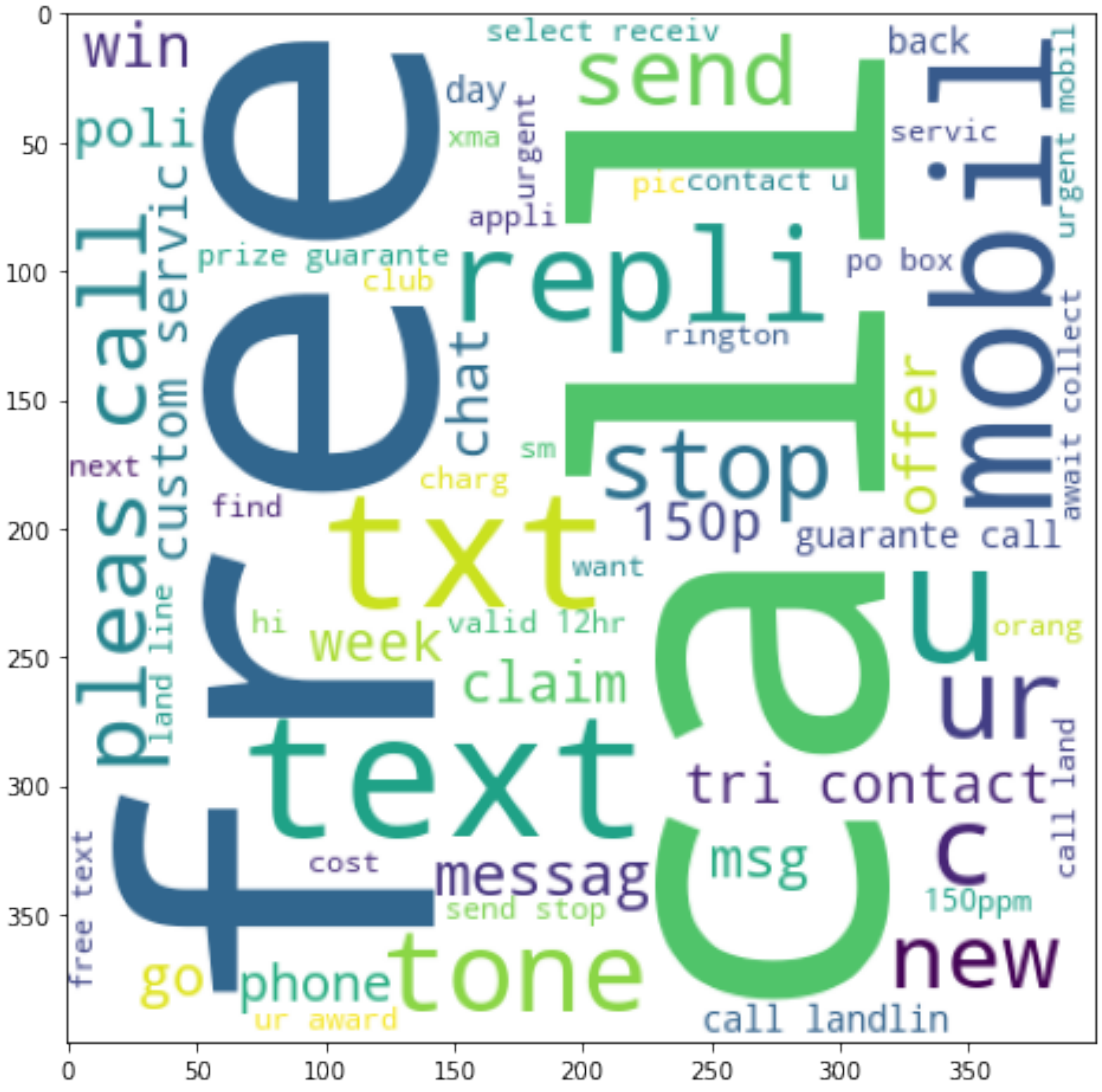
4 4. Data Visualization

[44]:
```
from wordcloud import WordCloud
wc = WordCloud(height = 400, background_color='white',min_font_size=10)
```

[45]:
```
wc_spam = wc.generate(df[df['target']==1]['transformed_text'].str.cat(sep=' '))
```

[46]:
```
plt.figure(figsize=(8,8))
plt.imshow(wc_spam)
```

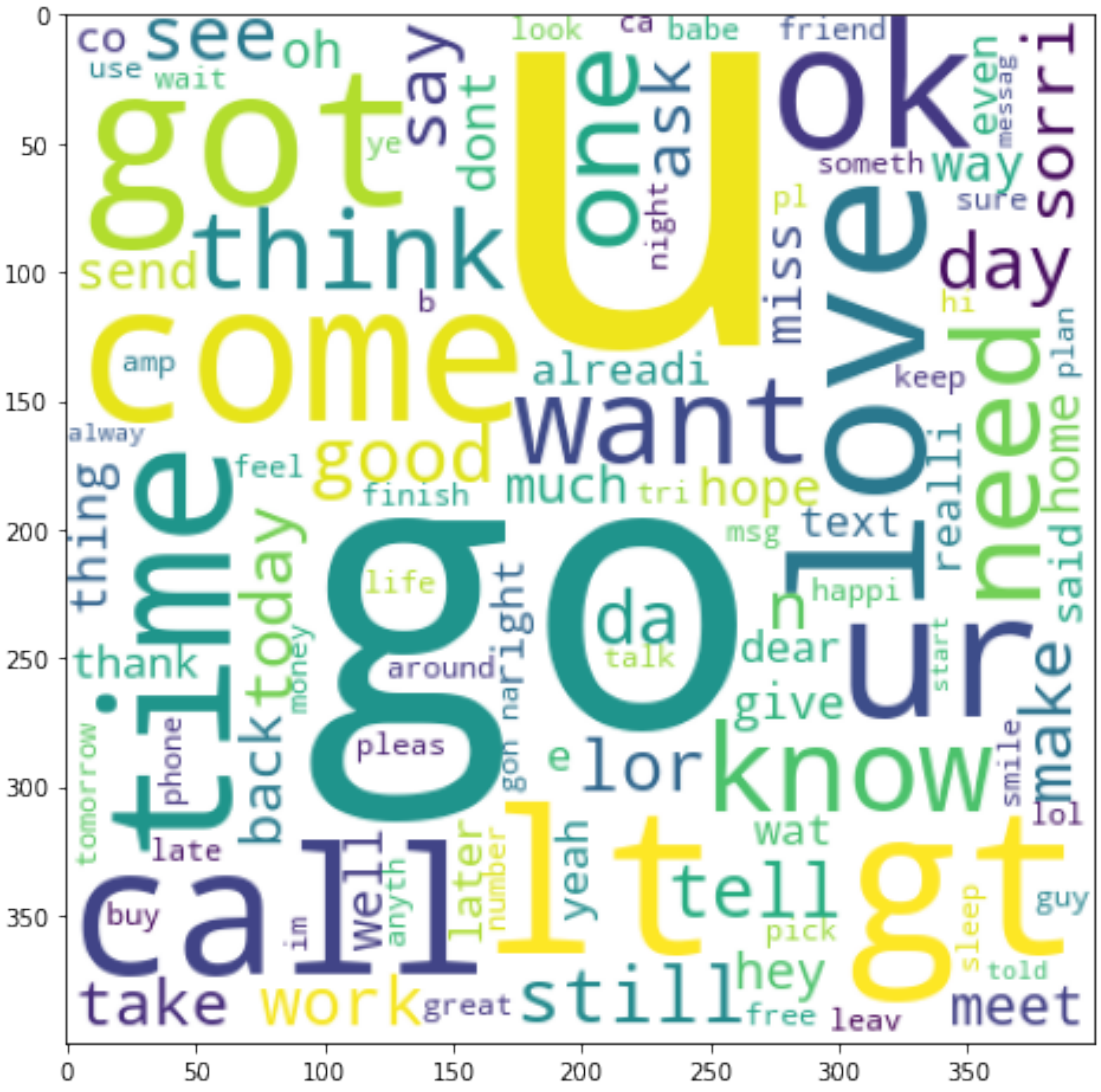
[46]: <matplotlib.image.AxesImage at 0x1e8c3690ca0>



[47]:
```
wc_ham = wc.generate(df[df['target']==0]['transformed_text'].str.cat(sep=' '))
```

[48]:
```
plt.figure(figsize=(8,8))
plt.imshow(wc_ham)
```

[48]: <matplotlib.image.AxesImage at 0x1e8c3d427f0>



[49]:
```
#top 30 words used in spam messages
spam_corpus = []
for msg in df[df['target']==1]['transformed_text'].tolist():
    for word in msg.split():
        spam_corpus.append(word)
```

[50]:
```
len(spam_corpus)
```

[50]:
```
9941
```

[51]:
```
from collections import Counter
```

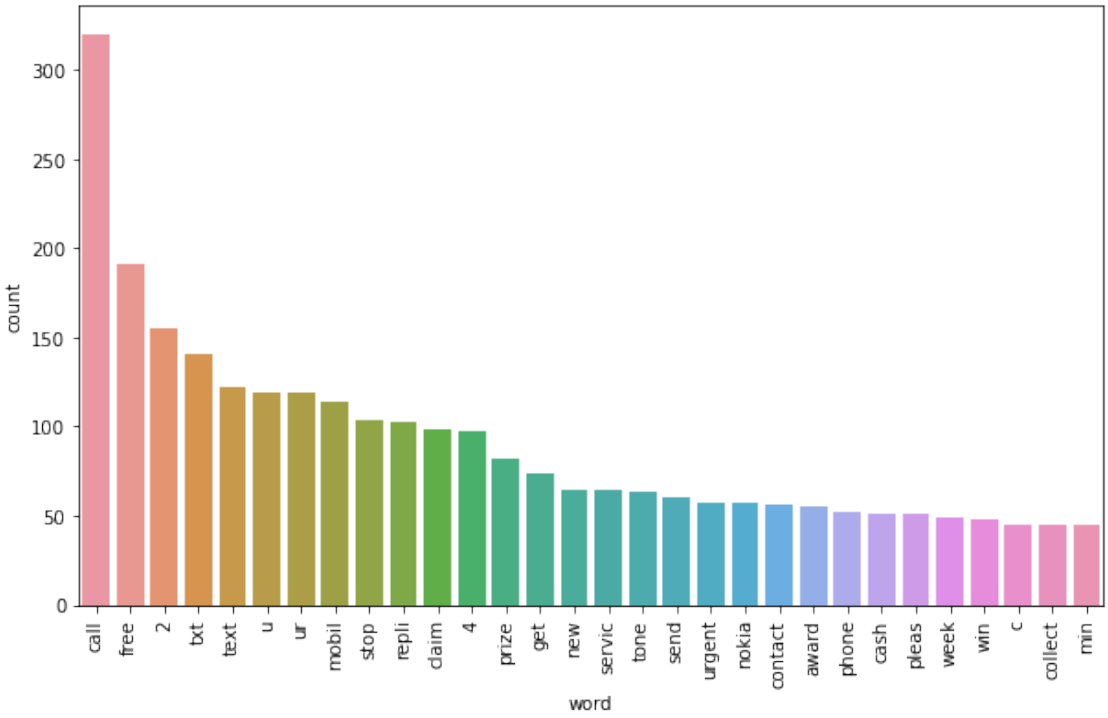
[52]:
```
common_spam_words = pd.DataFrame(Counter(spam_corpus).most_common(30))
```

[53]:
```
common_spam_words.columns
```

[53]:
```
RangeIndex(start=0, stop=2, step=1)
```

[54]:
```
common_spam_words.rename(columns={0:'word',1:'count'}, inplace=True)
```

[55]:
```
plt.figure(figsize=(10,6))
sns.barplot(x='word',y='count',data=common_spam_words)
plt.xticks(rotation='vertical')
plt.show()
```

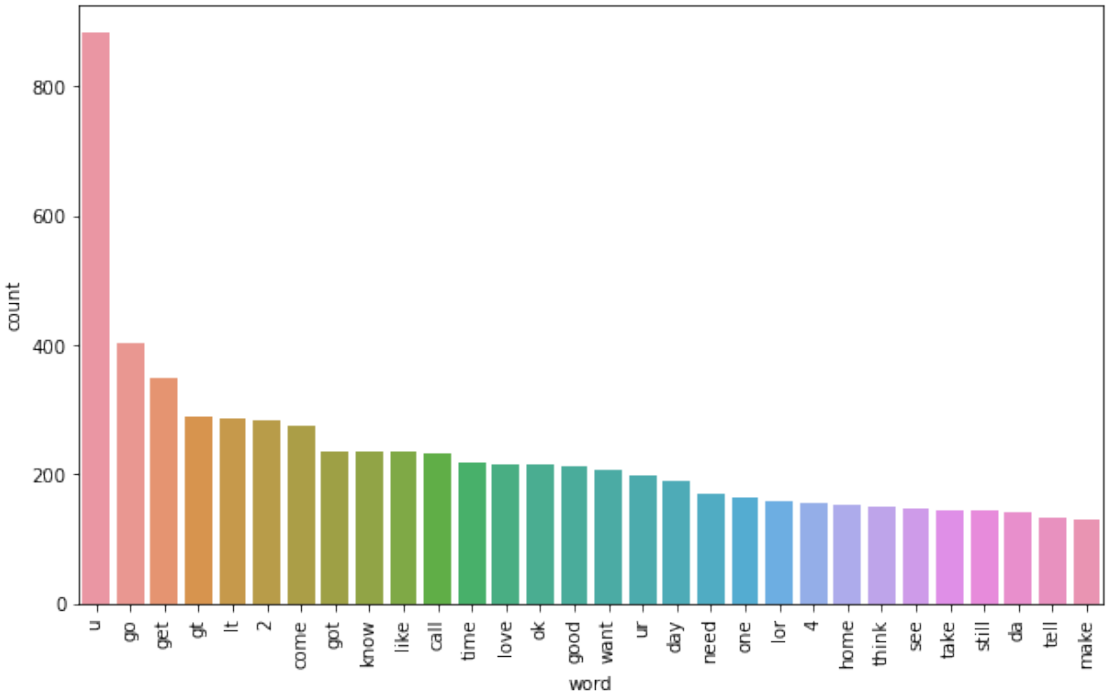


[56]:
```
# top 30 words used in ham msgs
ham_corpus = []
for msg in df[df['target']==0]['transformed_text'].tolist():
    for word in msg.split():
        ham_corpus.append(word)
```

[57]:
```
common_ham_words = pd.DataFrame(Counter(ham_corpus).most_common(30))
```

[58]:
```
common_ham_words.rename(columns={0:'word',1:'count'},inplace=True)
```

[59]:
```
plt.figure(figsize=(10,6))
sns.barplot(x='word',y='count',data=common_ham_words)
plt.xticks(rotation='vertical')
plt.show()
```



5 5. Model Building

[60]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

[61]:
```
cv = CountVectorizer()
```

[62]:
```
X = cv.fit_transform(df['transformed_text']).toarray()
```

[63]:
```
X.shape
```

[63]:
```
(5169, 6677)
```

[64]:
```
y = df['target'].values
```

[65]:
```
y
```

[65]:
```
array([0, 0, 1, ..., 0, 0, 0])
```

[66]:
```
from sklearn.model_selection import train_test_split
```

[76]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

[68]:
```
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
```

[69]:
```
gnb = GaussianNB()
mnb = MultinomialNB()
bnb = BernoulliNB()
```

[77]:
```
gnb.fit(X_train,y_train)
gnb_pred = gnb.predict(X_test)

mnb.fit(X_train,y_train)
mnb_pred = mnb.predict(X_test)

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

[71]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
```

[78]:
```
#gaussian Naive Bayes Performance
print(accuracy_score(y_test, gnb_pred))
print(confusion_matrix(y_test, gnb_pred))
print(precision_score(y_test,gnb_pred))
```

```
0.8762088974854932
[[783 109]
 [ 19 123]]
0.5301724137931034
```

[79]:
```
#Multinomial Naive Bayes Performance
print(accuracy_score(y_test, mnb_pred))
print(confusion_matrix(y_test, mnb_pred))
print(precision_score(y_test,mnb_pred))
```

```
0.9700193423597679
[[875  17]
 [ 14 128]]
0.8827586206896552
```

[80]:
```
#Bernoulli Naive Bayes Performance
print(accuracy_score(y_test, bnb_pred))
print(confusion_matrix(y_test, bnb_pred))
print(precision_score(y_test,bnb_pred))
```

```
0.9632495164410058
[[889   3]
 [ 35 107]]
0.9727272727272728
```

Bernoulli NB performed best among the three models.

6 Using Tf-Idf

[81]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
```

[104]:
```
X = tfidf.fit_transform(df['transformed_text']).toarray()
```

[105]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

[84]:
```
g = GaussianNB()
m = MultinomialNB()
b = BernoulliNB()
```

[106]:
```
g.fit(X_train,y_train)
pred1 = g.predict(X_test)
print(accuracy_score(y_test, pred1))
print(confusion_matrix(y_test, pred1))
print(precision_score(y_test,pred1))
```

```
0.8597678916827853
[[775 125]
 [ 20 114]]
0.4769874476987448
```

[107]:
```
m.fit(X_train,y_train)
pred2 = m.predict(X_test)
print(accuracy_score(y_test, pred2))
print(confusion_matrix(y_test, pred2))
print(precision_score(y_test,pred2))
```

```
0.9564796905222437
[[900   0]
 [ 45  89]]
1.0
```

[108]:
```
b.fit(X_train,y_train)
pred3 = b.predict(X_test)
print(accuracy_score(y_test, pred3))
print(confusion_matrix(y_test, pred3))
print(precision_score(y_test,pred3))
```

```
0.9690522243713733
[[895   5]
 [ 27 107]]
0.9553571428571429
```

Using Tf-Idf gave us even better performance with Multinomial NB.

7 Improving Our Model Performance

We can use different parameters in our model to improve the performance

[111]:
```
tf_idf = TfidfVectorizer(max_features=3000)
```

[112]:
```
X = tf_idf.fit_transform(df['transformed_text']).toarray()
```

[113]:
```
y = df['target'].values
```

[114]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

[92]:
```
gaussian = GaussianNB()
multinb = MultinomialNB()
bernoulli = BernoulliNB()
```

[115]:
```
#gaussian NB Evaluation
gaussian.fit(X_train,y_train)
pred1 = gaussian.predict(X_test)
print(accuracy_score(y_test,pred1))
print(confusion_matrix(y_test,pred1))
print(precision_score(y_test,pred1))
```

```
0.8820116054158608
[[804 103]
 [ 19 108]]
0.5118483412322274
```

[116]:
```
#Multinomial NB Evaluation
multinb.fit(X_train,y_train)
pred2 = multinb.predict(X_test)
print(accuracy_score(y_test,pred2))
print(confusion_matrix(y_test,pred2))
print(precision_score(y_test,pred2))
```

```
0.97678916827853
[[907   0]
 [ 24 103]]
1.0
```

[117]:
```
#Bernoulli NB Evaluation
bernoulli.fit(X_train,y_train)
```

```
pred3 = bernoulli.predict(X_test)
print(accuracy_score(y_test,pred3))
print(confusion_matrix(y_test,pred3))
print(precision_score(y_test,pred3))
```

```
0.9816247582205029
[[907   0]
 [ 19 108]]
1.0
```

Bernoulli Naive Bayes Model is working pretty well. We will use this classifier to make a Web App.

[109]:
```
import pickle
```

[118]:
```
pickle.dump(tf_idf,open('tfidf_vectorizer.pkl','wb'))
pickle.dump(bernoulli,open('final_model.pkl','wb'))
```

[ ]:
```

```