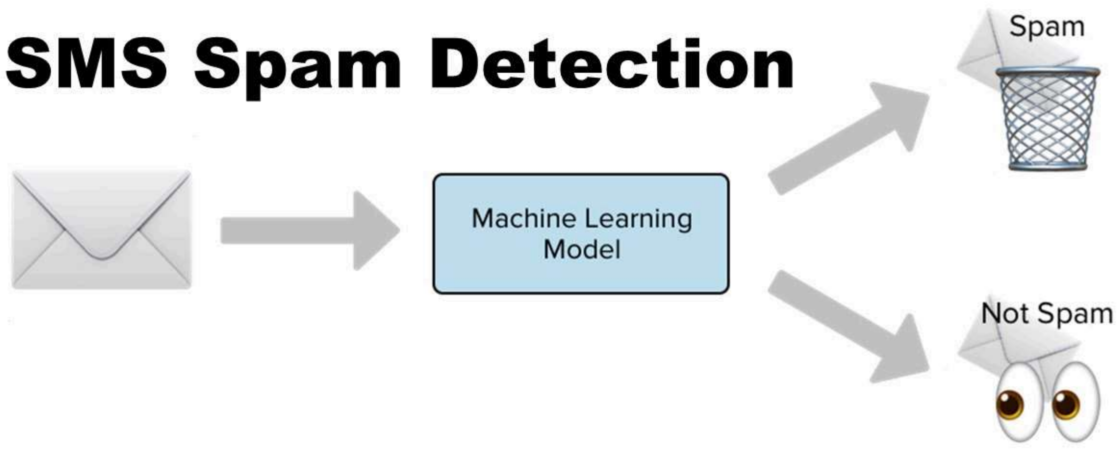
# Spam_SMS_email_Classifier

***-Mayank Srivastava***

LINKEDIN CONNECT

- [(https://linkedin.com/in/mayank-srivastava-6a8421105/)](https://linkedin.com/in/mayank-srivastava-6a8421105/)




Dataset: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset (https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset)

```
# 1. Data cleaning
# 2. EDA
# 3. Text Preprocessing
# 4. Model building
# 5. Evaluation
# 6. Improvement
# 7. Website
# 8. Deploy
```

In [1]:

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

In [2]:

```python
df= pd.read_csv('spam.csv', encoding='latin-1')
df.head()
```

Out[2]:

| | 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 |

In [3]:

```
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
```

```
Observations:

- No Null values
- size  = 5572 x 4
```

# 1. Data Cleaning

In [4]:

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

#renaming the columns
df.rename(columns ={'v1': 'label', 'v2': 'text'}, inplace = True)

# replacing ham ->0 , spam -> 1
df.label.replace(['ham','spam'],[0,1], inplace = True)

df.head()
```

Out[4]:

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

In [5]:

```
# checking for dupplicates

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

Out[5]: 403

In [6]:

```
# removing duplicates
df =df[~df.duplicated()]
df.shape
```

Out[6]: (5169, 2)

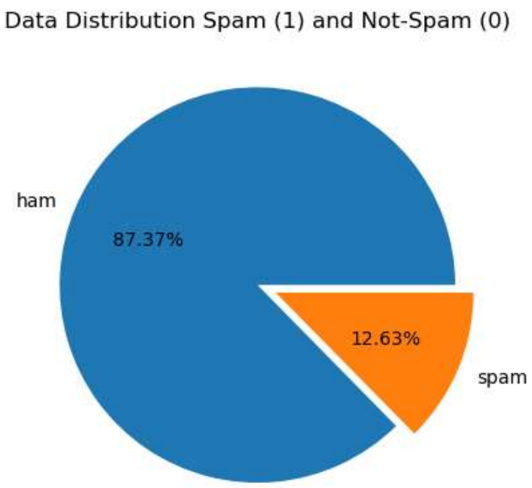
In [7]:

```
5572-403
```

Out[7]: 5169

## 2. EDA

In [8]:
```python
data= df.label.value_counts()
plt.pie(data.values, labels=['ham', 'spam'], autopct ="%1.2f%%", explode = [0.0,0.1])
plt.title('Data Distribution Spam (1) and Not-Spam (0)')
plt.show()
data
```




Out[8]:
```
label
0    4516
1     653
Name: count, dtype: int64
```

**Feature Engineering**

In [9]:
```python
# lets find no. of characters, words and sentences in spam and ham
import nltk
from nltk import word_tokenize,sent_tokenize

df['n_chars'] = df.text.str.len()
df['n_words'] =df.text.apply(lambda x: len(word_tokenize(x)))
df['n_sent'] =df.text.apply(lambda x: len(sent_tokenize(x)))
df.head()
```

Out[9]:

| | label | text | n_chars | n_words | n_sent |
|---|---|---|---|---|---|
| **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 |

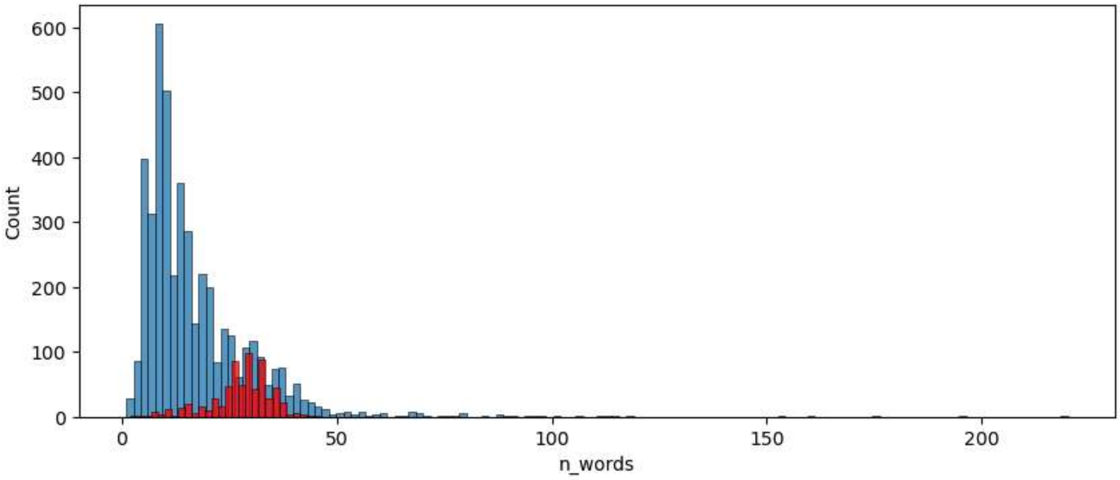
In [10]:
```python
# plotting spam vs ham based on no. of characters
plt.figure(figsize =(10,4))
sns.histplot(df[df.label == 0]['n_chars'])
sns.histplot(df[df.label == 1]['n_chars'], color= 'red')
```

Out[10]: `<Axes: xlabel='n_chars', ylabel='Count'>`

Observation: Spam text has generally higher character count than ham

In [11]:
```
# plotting spam vs ham based on no. of words
plt.figure(figsize =(10,4))
sns.histplot(df[df.label == 0]['n_words'])
sns.histplot(df[df.label == 1]['n_words'], color= 'red')
```

Out[11]:
```
<Axes: xlabel='n_words', ylabel='Count'>
```




Observation: Spam text has generally higher word count than ham

In [12]:
```
df.groupby('label')[['n_chars','n_words','n_sent']].mean().transpose().rename(columns ={0:'ham', 1:'spam'})
```

Out[12]:

| label | ham | spam |
|---|---|---|
| n_chars | 70.459256 | 137.891271 |
| n_words | 17.123782 | 27.667688 |
| n_sent | 1.820195 | 2.970904 |

Observation- On an average:
- ham text has avg 70 characters while spam has almost double characters.
- ham text has 17 words, spam has 27 words
- ham has ~ 2 sentences, spam has ~3 sentences.

## 3. Text preprocessing

In [13]:
```
import nltk

nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords

from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
```

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

In [14]:
```
# Lets define a helper fuction for standardization of text
```

In [15]:
```
# remove punctuations
import string
punctuation= string.punctuation
punctuation
```

Out[15]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [16]:

```
# stop words
from nltk.corpus import stopwords
stops= set(stopwords.words('english'))
stops
```

Out[16]:

```
{'a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'ain',
 'all',
 'am',
 'an',
 'and',
 'any',
 'are',
 'aren',
 "aren't",
 'as',
 'at',
 'be',
 'because',
```

In [17]:

```
import re
stem =PorterStemmer()
lemm = WordNetLemmatizer()
def standardize(text):

    # change to lower case
    text = text.lower()

    # keep only alpha numerics
    # assuming _ sign is used for space in text, replacing it with space
    text =re.sub('_','',text)
    text =re.findall(r"\w+", text)

    # now text has been converted into list of words , after re.findall

    # remove punctuations and stopwords
    text =[i for i in text if i not in stops and i not in punctuation]

    # lemmatization
    text= [lemm.lemmatize(i) for i in text]

    # stemming
    text= [stem.stem(i) for i in text]

    return(' '.join(text))
```

In [18]:

```
standardize("Hello' there I am peter, from22nd_street the mayor's office, who__ are. you? Dear %. Dancing ate remote
```

Out[18]:

```
'hello peter from22ndstreet mayor offic dear danc ate remot'
```

In [19]:

```
df['std_text'] = df.text.apply(standardize)
df.std_text
```

Out[19]:

```
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 go usf life around though
                              ...
5567    2nd time tri 2 contact u u å 750 pound prize 2...
5568                             ì b go esplanad fr home
5569                                   piti mood suggest
5570    guy bitch act like interest buy someth els nex...
5571                                      rofl true name
Name: std_text, Length: 5169, dtype: object
```

In [20]:

```
# pip install wordcloud
```

In [21]:

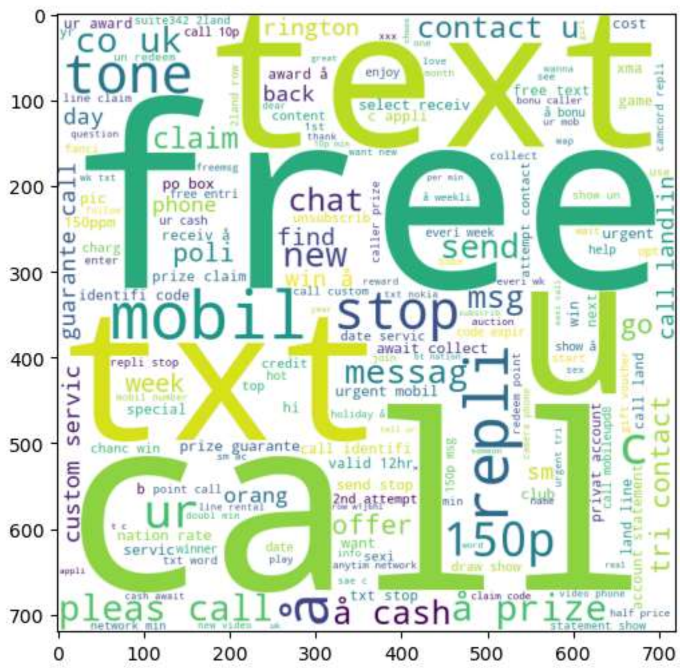
```
# word_cloud of spam messages
from wordcloud import WordCloud
wc= WordCloud(width =720, height =720, min_font_size =10, background_color ='white')
```

In [22]:

```
spam_wc = wc.generate(df[df.label == 1]['std_text'].astype('str').str.cat(sep= " "))
# concatentaing texts with seperator space
```

In [23]:
```python
plt.figure(figsize =(8,6))
plt.imshow(spam_wc)
```

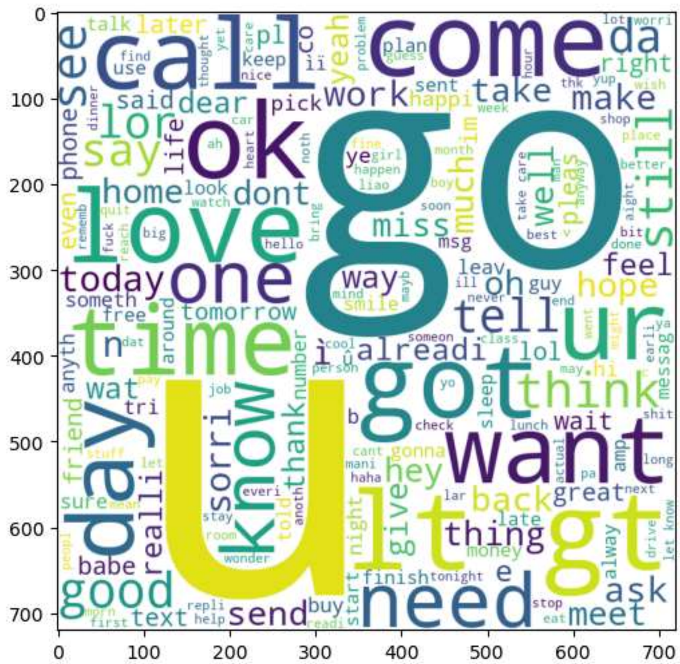
Out[23]: <matplotlib.image.AxesImage at 0x232ef89a810>




In [24]:
```python
ham_wc = wc.generate(df[df.label == 0]['std_text'].astype('str').str.cat(sep= " "))
# concatentaing texts with seperator space
```

In [25]:
```python
plt.figure(figsize =(8,6))
plt.imshow(ham_wc)
```

Out[25]: <matplotlib.image.AxesImage at 0x232ef9ba490>




In [26]:
```python
df[df.label == 1]['std_text']
```

Out[26]:
```
2       free entri 2 wkli comp win fa cup final tkt 21...
5       freemsg hey darl 3 week word back like fun sti...
8       winner valu network custom select receivea å 9...
9       mobil 11 month u r entitl updat latest colour ...
11      six chanc win cash 100 20 000 pound txt csh11 ...
                              ...
5537    want explicit sex 30 sec ring 02073162414 cost...
5540    ask 3mobil 0870 chatlin inclu free min india c...
5547    contract mobil 11 mnth latest motorola nokia e...
5566    remind o2 get 2 50 pound free call credit deta...
5567    2nd time tri 2 contact u u å 750 pound prize 2...
Name: std_text, Length: 653, dtype: object
```

In [27]:

```python
# Top 30 words in spam,
# made a single list, used value_counts
spam_words =[]
for i in df[df.label == 1]['std_text'].str.split(' '):
    spam_words+=i
print(spam_words)
```

```
'1', 'minmobsmorelkpobox177hp51fl', 'urgent', 'tri', 'contact', 'u', 'today', 'draw', 'show', 'å', '800', 'prize', 'g
uarante', 'call', '09050001295', 'land', 'line', 'claim', 'a21', 'valid', '12hr', 'monthli', 'password', 'wap', 'mobs
i', 'com', '391784', 'use', 'wap', 'phone', 'pc', 'today', 'vodafon', 'number', 'end', '0089', 'last', 'four', 'digi
t', 'select', 'receiv', 'å', '350', 'award', 'number', 'match', 'pleas', 'call', '09063442151', 'claim', 'å', '350',
'award', 'free', 'top', 'rington', 'sub', 'weekli', 'rington', 'get', '1st', 'week', 'free', 'send', 'subpoli', '8161
8', '3', 'per', 'week', 'stop', 'sm', '08718727870', 'free', 'msg', 'sorri', 'servic', 'order', '81303', 'could', 'de
liv', 'suffici', 'credit', 'pleas', 'top', 'receiv', 'servic', 'hard', 'live', '121', 'chat', '60p', 'min', 'choos',
'girl', 'connect', 'live', 'call', '09094646899', 'cheap', 'chat', 'uk', 'biggest', 'live', 'servic', 'vu', 'bcm1896w
c1n3xx', 'wow', 'boy', 'r', 'back', 'take', '2007', 'uk', 'tour', 'win', 'vip', 'ticket', 'pre', 'book', 'vip', 'clu
b', 'txt', 'club', '81303', 'trackmarqu', 'ltd', 'info', 'vipclub4u', 'hi', 'mandi', 'sullivan', 'call', 'hotmix', 'f
m', 'chosen', 'receiv', 'å', '5000', '00', 'easter', 'prize', 'draw', 'pleas', 'telephon', '09041940223', 'claim', '2
9', '03', '05', 'prize', 'transfer', 'someon', 'els', 'ur', 'go', '2', 'bahama', 'callfreefon', '08081560665', 'spea
k', 'live', 'oper', 'claim', 'either', 'bahama', 'cruis', 'ofå', '2000', 'cash', '18', 'opt', 'txt', 'x', '0778620011
7', 'someon', 'conact', 'date', 'servic', 'enter', 'phone', 'fanci', 'find', 'call', 'landlin', '09111030116', 'pobox
12n146tf15', 'hi', '07734396839', 'ibh', 'custom', 'loyalti', 'offer', 'new', 'nokia6600', 'mobil', 'å', '10', 'txtau
ction', 'txt', 'word', 'start', '81151', 'get', '4t', 'sm', 'auction', 'nokia', '7250i', 'get', 'win', 'free', 'aucti
on', 'take', 'part', 'send', 'nokia', '86021', 'hg', 'suite342', '2land', 'row', 'w1jhl', '16', 'call', 'freephon',
'0800', '542', '0578', 'buy', 'space', 'invad', '4', 'chanc', '2', 'win', 'orig', 'arcad', 'game', 'consol', 'press',
'0', 'game', 'arcad', 'std', 'wap', 'charg', 'see', 'o2', 'co', 'uk', 'game', '4', 'term', 'set', 'purchas', 'big',
'brother', 'alert', 'comput', 'select', 'u', '10k', 'cash', '150', 'voucher', 'call', '09064018838', 'ntt', 'po', 'bo
```

In [28]:

```python
len(spam_words)
```

Out[28]:

```
11996
```

In [29]:

```python
from collections import Counter
spam_count =Counter(spam_words)
spam_count.most_common(30)
```

Out[29]:

```
[('call', 327),
 ('å', 247),
 ('free', 195),
 ('2', 184),
 ('u', 155),
 ('txt', 145),
 ('text', 128),
 ('ur', 119),
 ('mobil', 118),
 ('4', 114),
 ('stop', 109),
 ('repli', 103),
 ('claim', 98),
 ('1', 93),
 ('c', 87),
 ('prize', 83),
 ('www', 83),
 ('get', 75),
 ('min', 71),
 ('tone', 70),
 ('cash', 65),
 ('servic', 65),
 ('150p', 64),
 ('new', 64),
 ('uk', 63),
 ('send', 61),
 ('nokia', 59),
 ('urgent', 58),
 ('msg', 58),
 ('contact', 56)]
```

In [30]:

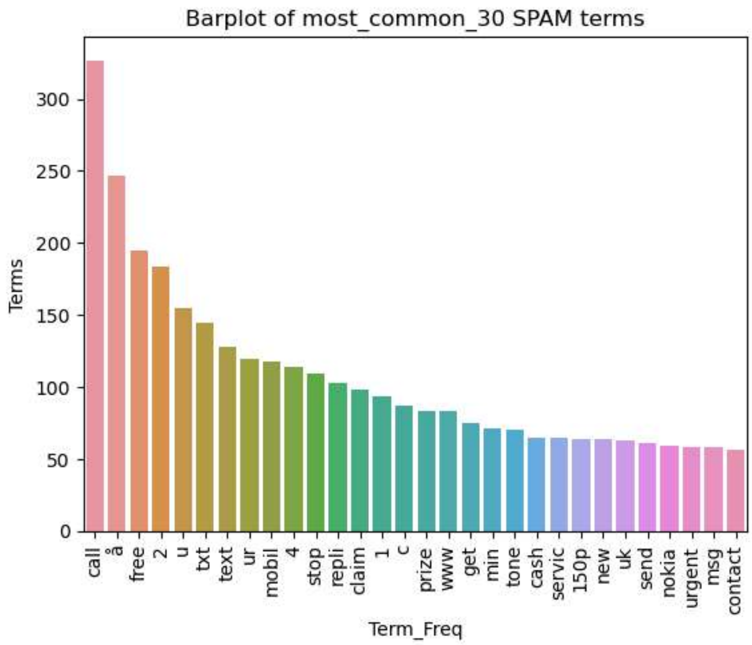
```
pd.DataFrame(spam_count.most_common(30))
```

Out[30]:

| | 0 | 1 |
|---|---|---|
| **0** | call | 327 |
| **1** | à | 247 |
| **2** | free | 195 |
| **3** | 2 | 184 |
| **4** | u | 155 |
| **5** | txt | 145 |
| **6** | text | 128 |
| **7** | ur | 119 |
| **8** | mobil | 118 |
| **9** | 4 | 114 |
| **10** | stop | 109 |
| **11** | repli | 103 |
| **12** | claim | 98 |
| **13** | 1 | 93 |
| **14** | c | 87 |
| **15** | prize | 83 |
| **16** | www | 83 |
| **17** | get | 75 |
| **18** | min | 71 |
| **19** | tone | 70 |
| **20** | cash | 65 |
| **21** | servic | 65 |
| **22** | 150p | 64 |
| **23** | new | 64 |
| **24** | uk | 63 |
| **25** | send | 61 |
| **26** | nokia | 59 |
| **27** | urgent | 58 |
| **28** | msg | 58 |
| **29** | contact | 56 |

In [31]:

```
sns.barplot(x= pd.DataFrame(spam_count.most_common(30))[0],y= pd.DataFrame(spam_count.most_common(30))[1])
plt.xticks(rotation ='vertical')
plt.xlabel('Term_Freq')
plt.ylabel('Terms')
plt.title('Barplot of most_common_30 SPAM terms')
plt.show()
```



In [32]:

```python
# similarly for ham_words
ham_words = []
for i in df[df.label == 0]['std_text'].str.split():
    ham_words+=i
print(ham_words)
```

```
'soon', 'come', 'otherwis', 'tomorrow', 'msg', 'r', 'time', 'pa', 'silent', 'say', 'think', 'u', 'right', 'also', 'ma
ke', 'u', 'think', 'least', '4', 'moment', 'gd', 'nt', 'swt', 'drm', 'shesil', 'yeah', 'probabl', 'swing', 'roommat',
'finish', 'girl', 'happi', 'new', 'year', 'melodi', 'ìï', 'dun', 'need', 'pick', 'ur', 'gf', 'yay', 'better', 'told',
'5', 'girl', 'either', 'horribl', 'u', 'eat', 'mac', 'eat', 'u', 'forgot', 'abt', 'alreadi', 'rite', 'u', 'take', 'lo
ng', '2', 'repli', 'thk', 'toot', 'b4', 'b', 'prepar', 'wat', 'shall', 'eat', 'say', 'fantast', 'chanc', 'anyth', 'ne
ed', 'bigger', 'life', 'lift', 'lose', '2', 'live', 'think', 'would', 'first', 'person', '2', 'die', 'n', 'v', 'q',
'nw', 'came', 'hme', 'da', 'outsid', 'island', 'head', 'toward', 'hard', 'rock', 'run', 'day', 'class', 'class', 'che
nnai', 'velacheri', 'flippin', 'shit', 'yet', 'k', 'give', 'sec', 'break', 'lt', 'gt', 'cstore', 'much', 'bad', 'avoi
d', 'like', 'yo', 'around', 'got', 'car', 'back', 'annoy', 'goodmorn', 'today', 'late', 'lt', 'gt', 'min', 'point',
'hangin', 'mr', 'right', 'makin', 'u', 'happi', 'come', 'aliv', 'better', 'correct', 'good', 'look', 'fi
gur', 'case', 'guess', 'see', 'campu', 'lodg', 'done', 'come', 'home', 'one', 'last', 'time', 'wont', 'anyth', 'trus
t', 'night', 'worri', 'appt', 'shame', 'miss', 'girl', 'night', 'quiz', 'popcorn', 'hair', 'ok', 'c', 'ya', 'said',
'matter', 'mind', 'say', 'matter', 'al', 'moan', 'n', 'e', 'thin', 'go', 'wrong', 'fault', 'al', 'de', 'argument',
'r', 'fault', 'fed', 'himso', 'bother', 'hav', '2go', 'thanx', 'xx', 'neft', 'transact', 'refer', 'number', 'lt', 'g
t', 'r', 'lt', 'decim', 'gt', 'credit', 'beneficiari', 'account', 'lt', 'gt', 'lt', 'time', 'gt', 'lt', 'gt', 'otherw
is', 'part', 'time', 'job', 'na', 'tuition', 'know', 'call', 'also', 'da', 'feel', 'yesterday', 'night', 'wait', 'ti
l', '2day', 'night', 'dear', 'thank', 'understand', 'tri', 'tell', 'sura', 'whole', 'car', 'appreci', 'last', 'two',
'dad', 'map', 'read', 'semi', 'argument', 'apart', 'thing', 'go', 'ok', 'p', 'need', 'strong', 'arm', 'also', 'maaaa
n', 'miss', 'bday', 'real', 'april', 'guessin', 'gonna', '9', 'ok', 'come', 'ur', 'home', 'half', 'hour', 'yo', 'gam
e', 'almost', 'want', 'go', 'walmart', 'soon', 'yeah', 'probabl', 'sure', 'ilol', 'let', 'u', 'know', 'person', 'wuld
```

In [33]:

```python
len(ham_words)
```

Out[33]:

```
36505
```

In [34]:

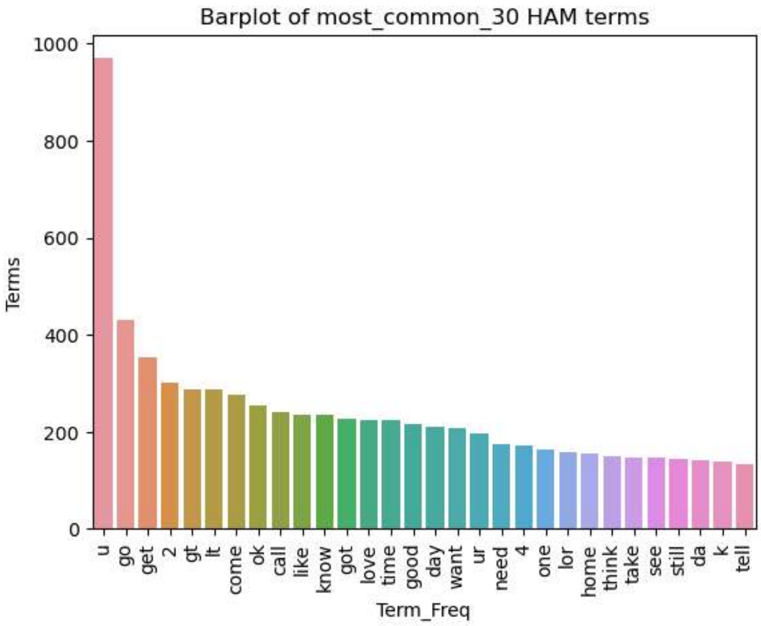
```python
from collections import Counter
ham_count =Counter(ham_words)
ham_count.most_common(30)
```

Out[34]:

```
[('u', 969),
 ('go', 432),
 ('get', 354),
 ('2', 302),
 ('gt', 288),
 ('lt', 287),
 ('come', 276),
 ('ok', 255),
 ('call', 240),
 ('like', 236),
 ('know', 236),
 ('got', 226),
 ('love', 225),
 ('time', 223),
 ('good', 216),
 ('day', 210),
 ('want', 209),
 ('ur', 198),
 ('need', 174),
 ('4', 171),
 ('one', 165),
 ('lor', 159),
 ('home', 156),
 ('think', 150),
 ('take', 148),
 ('see', 147),
 ('still', 145),
 ('da', 143),
 ('k', 138),
 ('tell', 134)]
```

In [35]:

```python
sns.barplot(x= pd.DataFrame(ham_count.most_common(30))[0],y= pd.DataFrame(ham_count.most_common(30))[1])
plt.xticks(rotation ='vertical')
plt.xlabel('Term_Freq')
plt.ylabel('Terms')
plt.title('Barplot of most_common_30 HAM terms')
plt.show()
```




## 4. Text Vectorization

In [36]:

```python
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
cv= CountVectorizer(stop_words = 'english')
tv= TfidfVectorizer(stop_words = 'english')

X_cv= cv.fit_transform(df.std_text).toarray()
X_tv= tv.fit_transform(df.std_text).toarray()
```

In [37]:

```python
X_cv.shape, X_tv.shape
```

Out[37]: ((5169, 7059), (5169, 7059))

- Both the vectorization methods give text vetors of same length

In [38]:

```python
y=df.label
```

In [39]:

```python
#Tf-idf  vectorization
from sklearn.model_selection import cross_val_score, train_test_split
xtrain, xtest, ytrain, ytest =train_test_split(X_tv, y, test_size =0.2, random_state =42)
```

## 5. Model Builidng

- Which is Better for Spam Detection?
  - If minimizing false positives is critical (i.e., you don't want to risk important emails being marked as spam), precision is more important.
  - This is often the case in professional or business environments where missing an important email can have serious consequences.
  - If minimizing false negatives is critical (i.e., you want to ensure that most spam emails are correctly identified), recall is more important.
  - This is often the case in personal email use where receiving spam emails is more of an annoyance than missing an important email.

In [40]:

```python
# to check names of scorers for cross-val
# import sklearn
# sklearn.metrics.get_scorer_names()
```

In [41]:

```python
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.model_selection import cross_val_score, train_test_split
gnb= GaussianNB()
mnb=MultinomialNB()
bnb =BernoulliNB()
```

In [42]:

```python
X={'BoW':X_cv, 'Tf-Idf':X_tv}
model={'GaussianNB':gnb,'MultinomialNB':mnb,'BernoulliNB':bnb}
```

In [43]:

```python
scores={}
for i in X:
    for j in model:
        scores[i+'_'+j] = np.max(cross_val_score(model[j], X[i], y, cv=5, scoring ='precision'))

# our metric is recall score because, we dont want any false positives (Type 2 error), ie a Spam predicted as Ham
```

In [44]:

```python
scores
```

Out[44]:

```
{'BoW_GaussianNB': 0.5041322314049587,
 'BoW_MultinomialNB': 0.9333333333333333,
 'BoW_BernoulliNB': 1.0,
 'Tf-Idf_GaussianNB': 0.497907949790795,
 'Tf-Idf_MultinomialNB': 1.0,
 'Tf-Idf_BernoulliNB': 1.0}
```

In [45]:

```python
sorted(scores.items(), key=lambda x: x[1], reverse = True)
```

Out[45]:

```
[('BoW_BernoulliNB', 1.0),
 ('Tf-Idf_MultinomialNB', 1.0),
 ('Tf-Idf_BernoulliNB', 1.0),
 ('BoW_MultinomialNB', 0.9333333333333333),
 ('BoW_GaussianNB', 0.5041322314049587),
 ('Tf-Idf_GaussianNB', 0.497907949790795)]
```

- Clearly the winner is BoW with Multinomial Naive Bayes with the best f1_score

In [46]:

```python
# training the model on 'BoW_BernoulliNB' or 'Tf-Idf_MultinomialNB' or 'Tf-Idf_BernoulliNB' is giving ideal Precisio

# randomly proceeding with Tf-Idf_MultinomialNB
xtrain, xtest, ytrain, ytest =train_test_split(X_tv, y, test_size =0.2, random_state =42)
mnb.fit(xtrain, ytrain)
```

Out[46]:

```
▾ MultinomialNB
MultinomialNB()
```

In [47]:

```python
from sklearn.metrics import accuracy_score, precision_score, classification_report, confusion_matrix
pred_train =mnb.predict(xtrain)
pred_test =mnb.predict(xtest)
print('training accuracy_score: ', accuracy_score(ytrain, pred_train))
print('training precision_score: ', precision_score(ytrain, pred_train))
print('training confusion_matrix: \n', confusion_matrix(ytrain, pred_train))

#predicting testing data
print('testing accuracy_score: ', accuracy_score(ytest, pred_test))
print('testing precision_score: ', precision_score(ytest, pred_test))
print('testing confusion_matrix: \n', confusion_matrix(ytest, pred_test))
```

```
training accuracy_score:  0.9743651753325272
training precision_score:  1.0
training confusion_matrix: 
 [[3627    0]
 [ 106  402]]
testing accuracy_score:  0.9661508704061895
testing precision_score:  1.0
testing confusion_matrix: 
 [[889   0]
 [ 35 110]]
```

In [48]:

```python
# 'Tf-Idf_BernoulliNB'
xtrain, xtest, ytrain, ytest =train_test_split(X_tv, y, test_size =0.2, random_state =42)
bnb.fit(xtrain, ytrain)
from sklearn.metrics import accuracy_score, precision_score, classification_report, confusion_matrix
pred_train =bnb.predict(xtrain)
pred_test =bnb.predict(xtest)
print('training accuracy_score: ', accuracy_score(ytrain, pred_train))
print('training precision_score: ', precision_score(ytrain, pred_train))
print('training confusion_matrix: \n', confusion_matrix(ytrain, pred_train))

#predicting testing data
print('testing accuracy_score: ', accuracy_score(ytest, pred_test))
print('testing precision_score: ', precision_score(ytest, pred_test))
print('testing confusion_matrix: \n', confusion_matrix(ytest, pred_test))
```

```
training accuracy_score:  0.9835550181378476
training precision_score:  0.9932735426008968
training confusion_matrix: 
 [[3624    3]
 [  65  443]]
testing accuracy_score:  0.971953578336557
testing precision_score:  0.967741935483871
testing confusion_matrix: 
 [[885   4]
 [ 25 120]]
```

In [49]:

```python
# 'BoW_BernoulliNB'
xtrain, xtest, ytrain, ytest =train_test_split(X_cv, y, test_size =0.2, random_state =42)
bnb.fit(xtrain, ytrain)
from sklearn.metrics import accuracy_score, precision_score, classification_report, confusion_matrix
pred_train =bnb.predict(xtrain)
pred_test =bnb.predict(xtest)
print('training accuracy_score: ', accuracy_score(ytrain, pred_train))
print('training precision_score: ', precision_score(ytrain, pred_train))
print('training confusion_matrix: \n', confusion_matrix(ytrain, pred_train))

#predicting testing data
print('testing accuracy_score: ', accuracy_score(ytest, pred_test))
print('testing precision_score: ', precision_score(ytest, pred_test))
print('testing confusion_matrix: \n', confusion_matrix(ytest, pred_test))
```

```
training accuracy_score:  0.9835550181378476
training precision_score:  0.9932735426008968
training confusion_matrix:
 [[3624    3]
 [  65  443]]
testing accuracy_score:  0.971953578336557
testing precision_score:  0.967741935483871
testing confusion_matrix:
 [[885   4]
 [ 25 120]]
```

- Since for a Business case **'precision'** is more important than **'accuracy'** for a Spam classifier, we will go ahead with **Tf-Idf-MultinomialNB.**
- Achieved Precision: 1.0
- Accuracy 0.9661508704061895

## 6. Model Improvement

**Trying other ML classifer algorithm**

In [50]:

```python
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, AdaBoostClassifier, BaggingClassifier, ExtraTreesClassifier,Gra
from xgboost import XGBClassifier
```

In [51]:

```python
lr=LogisticRegression()
svc =SVC()
mnb=MultinomialNB()
dt= DecisionTreeClassifier()
knn=KNeighborsClassifier()
rf= RandomForestClassifier()
ada= AdaBoostClassifier()
bg=BaggingClassifier()
etc= ExtraTreesClassifier()
gb= GradientBoostingClassifier()
xgb= XGBClassifier()
```

In [52]:

```python
models ={'LR': lr, 'SVC' :svc, 'MultNB':mnb, 'DT':dt, "KNN":knn,"RF":rf, "ADA":ada,"BgC":bg,"ETC":etc,"GB":gb,"XGB":
```

In [53]:

```python
xtrain, xtest, ytrain, ytest =train_test_split(X_tv, y, test_size =0.2, random_state =42)

def training_model(model, xtrain, xtest, ytrain, ytest):
    model.fit(xtrain,ytrain)
    pred= model.predict(xtest)
    accuracy =accuracy_score(ytest,pred)
    precision =precision_score(ytest,pred)

    return accuracy, precision
```

In [54]:

```python
accuracy=[]
precision =[]
for i in models:
    a,p=training_model(models[i],xtrain, xtest, ytrain, ytest)
    accuracy.append(a)
    precision.append(p)
```

In [55]:

```
final =pd.DataFrame({'Classifier' : models.keys(), 'Accuracy': accuracy, 'Precision': precision})
final.sort_values(by='Precision', ascending =False)
```

Out[55]:

| | Classifier | Accuracy | Precision |
|---|---|---|---|
| **2** | MultNB | 0.966151 | 1.000000 |
| **4** | KNN | 0.890716 | 1.000000 |
| **8** | ETC | 0.979691 | 0.992063 |
| **5** | RF | 0.977756 | 0.991935 |
| **1** | SVC | 0.970986 | 0.991453 |
| **9** | GB | 0.965184 | 0.990991 |
| **0** | LR | 0.954545 | 0.980392 |
| **10** | XGB | 0.972921 | 0.953488 |
| **6** | ADA | 0.974855 | 0.940741 |
| **7** | BgC | 0.967118 | 0.930233 |
| **3** | DT | 0.970019 | 0.895833 |

**Hypertuning the best model**

In [56]:

```
from sklearn.model_selection import GridSearchCV
```

In [57]:

```
from sklearn.pipeline import Pipeline
from sklearn.metrics import make_scorer
```

In [58]:

```
# hypertuning the TF-IDF -> MNB model

# Define a pipeline
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('nb', MultinomialNB())
])

# Define the parameter grid
param_grid = {
    'tfidf__max_df': [0.5, 0.75,1.0],
    'tfidf__min_df': [1, 2, 5],
    'tfidf__ngram_range': [(1, 1), (1, 2)],
    'nb__alpha': [0.01, 0.1, 1, 10]
}

# 'tfidf__max_df': Maximum document frequency (proportion of documents in which a term appears).
# 'tfidf__min_df': Minimum document frequency (minimum number of documents a term must be in).
# 'tfidf__ngram_range': The lower and upper boundary of the range of n-values for different n-grams to be extracted.
# 'nb__alpha': Smoothing parameter for the Naive Bayes classifier.

# Define the scoring dictionary
scoring = {
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score)
}
# Perform GridSearchCV
gv = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1, verbose=1, scoring = scoring, refit='accuracy')

# Fit the model
xtrain, xtest, ytrain, ytest =train_test_split(df.std_text.astype('str'), y, test_size =0.2, random_state =42)
gv.fit(xtrain, ytrain)

# Print best parameters and best score
print("Best parameters found: ", gv.best_params_)
print("Best accuracy_score: ", gv.best_score_)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Best parameters found:  {'nb__alpha': 0.1, 'tfidf__max_df': 0.5, 'tfidf__min_df': 2, 'tfidf__ngram_range': (1, 1)}
Best accuracy_score:  0.9842805320435308
```

In [59]:

```
# precision, accuracy
precision_score(ytest,gv.predict(xtest)), accuracy_score(ytest,gv.predict(xtest))
```

Out[59]:

```
(0.9692307692307692, 0.9777562862669246)
```

- the accuracy has improved from 0.0.9661508704061895 to 0.9777562862669246 , but precison has dropped from 1.0 to 0.9692307692307692
- this tradeoff is not desirable as precision holds greater significance for business use in a spam classifier
- so we will keep things unchanged an procced with the **TF-IDF-> MNB** as our final model

In [60]:
```python
X_tv= tv.fit_transform(df.std_text).toarray()
xtrain, xtest, ytrain, ytest =train_test_split(X_tv, y, test_size =0.2, random_state =42)
mnb.fit(xtrain, ytrain)
pred_train =mnb.predict(xtrain)
pred_test =mnb.predict(xtest)
print('testing accuracy_score: ', accuracy_score(ytest, pred_test))
print('testing precision_score: ', precision_score(ytest, pred_test))
print('testing confusion_matrix: \n', confusion_matrix(ytest, pred_test))
```

```
testing accuracy_score:  0.9661508704061895
testing precision_score:  1.0
testing confusion_matrix:
 [[889   0]
 [ 35 110]]
```

## 7. Pickle

In [61]:
```python
import pickle
pickle.dump(mnb, open('model.pkl', 'wb'))
pickle.dump(tv, open('vectorizer.pkl', 'wb'))
pickle.dump(standardize, open('standardize.pkl','wb'))
```

In [62]:
```python
xtest[9].shape
```

Out[62]: (7059,)

In [63]:
```python
xtest[0].reshape(1, -1).shape
```

Out[63]: (1, 7059)

In [64]:
```python
mnb.predict(xtest[5].reshape(1, -1)), ytest.iloc[5]
```

Out[64]: (array([1], dtype=int64), 1)

In [65]:
```python
mnb.predict(xtest[5].reshape(1, -1))[0]
```

Out[65]: 1

In [ ]:
```python

```

In [66]:
```python
t= """We’re happy to inform you that you’re entitled to a refund for overpayment on your AMEX account. Click on this
```

In [67]:
```python
standardize(t)
```

Out[67]: 'happi inform entitl refund overpay amex account click link link claim refund'

In [68]:
```python
tv.transform([standardize(t)]).shape
```

Out[68]: (1, 7059)

In [69]:
```python
mnb.predict(tv.transform([standardize(t)]))
```

Out[69]: array([0], dtype=int64)

In [70]:
```python
mnb.predict_proba(tv.transform([standardize(t)])).max()
```

Out[70]: 0.6703360228456645