In [29]:

```
import nltk
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.naive_bayes import MultinomialNB


from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline


from sklearn.metrics import classification_report, confusion_matrix


%matplotlib inline
```

Read the dataset

- from University of California Irvine Machine Learning Repository: UCI datasets (https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection)

In [2]:

```
messages = [line.rstrip() for line in open('SMSSpamCollection')]
```

In [3]:

```
# Check the first message

messages[0]
```

Out[3]:

```
'ham\tGo until jurong point, crazy.. Available only in bugis n great world
la e buffet... Cine there got amore wat...'
```

In [4]:

```
# Check the second message

messages[1]
```

Out[4]:

```
'ham\tOk lar... Joking wif u oni...'
```

In [5]:

```
# Check the last message

messages[-1]
```

Out[5]:

```
'ham\tRofl. Its true to its name'
```

In [6]:

```
# Length of messages

len(messages)
```

Out[6]:

```
5574
```

Note: Collection of text is sometimes called a 'corpus'

In [7]:

```
# Breakdown of collection of text or corpus

list1 = ['this is the place of encounter',
        'this is the place of surrneder',
        'do to me what you want Jesus']

for k, i in enumerate(list1):
    print(k, i)
```

```
0 this is the place of encounter
1 this is the place of surrneder
2 do to me what you want Jesus
```

Corpus of the first 7 messages in dataset

In [8]:

```
## ...alongside the message number, we have:

for message_no, message in enumerate(messages[:7]):
    print(message_no, message)
    print('\n')
```

```
0 ham	Go until jurong point, crazy.. Available only in bugis n great wor
ld la e buffet... Cine there got amore wat...


1 ham	Ok lar... Joking wif u oni...


2 spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 200
5. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 084
52810075over18's


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 around here though


5 spam	FreeMsg Hey there darling it's been 3 week's now and no word back!
I'd like some fun you up for it still? Tb ok! XxX std chgs to send, Â£1.50
to rcv


6 ham	Even my brother is not like to speak with me. They treat me like a
ids patent.
```

Note:

- From the 'Corpus' above, we can tell that the messages are separated with tabs, thus a 'tab-separated-value (tsv)' file.
- first column = label (ham or spam). ham and spam means 'good email' and 'bad email' respectively.
- second column = the message itself

In [9]:

```
# to further show that the 'messages' file is a tsv,
# let's check the first message


messages[0]
```

Out[9]:

```
'ham\tGo until jurong point, crazy.. Available only in bugis n great world
la e buffet... Cine there got amore wat...'
```

Observation:

- We see that the string above starts with " **ham\t** " meaning this is a 'good email', tab(**" \t "**) and then the message itself
- "Hence we will rather use -> pandas <- to work on the dataset rather than regular python"

Read the dataset

- as a 'csv' file

In [10]:

```
messages2 = pd.read_csv(filepath_or_buffer = 'SMSSpamCollection',
                        sep = '\t')
```

In [11]:

```
messages2



# To see the entire datafram

#pd.set_option("display.max_rows", None, "display.max_columns", None)
```

Out[11]:

	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
0	ham	Ok lar... Joking wif u oni...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...
2	ham	U dun say so early hor... U c already then say...
3	ham	Nah I don't think he goes to usf, he lives aro...
4	spam	FreeMsg Hey there darling it's been 3 week's n...
...	...	...
5566	spam	This is the 2nd time we have tried 2 contact u...
5567	ham	Will ü b going to esplanade fr home?
5568	ham	Pity, * was in mood for that. So...any other s...
5569	ham	The guy did some bitching but I acted like i'd...
5570	ham	Rofl. Its true to its name

5571 rows × 2 columns

Rename the columns:

In [12]:

```
messages2.columns = ['label', 'message']
```

In [13]:

```
messages2.head()


# this is a nice dataframe
```

Out[13]:

	label	message
0	ham	Ok lar... Joking wif u oni...
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...
2	ham	U dun say so early hor... U c already then say...
3	ham	Nah I don't think he goes to usf, he lives aro...
4	spam	FreeMsg Hey there darling it's been 3 week's n...

Exploratory Data Analysis

Statistical information

In [14]:

```
messages2.describe()
```

Out[14]:

	label	message
count	5571	5571
unique	2	5168
top	ham	Sorry, I'll call later
freq	4824	30

Observations:

- There are 5571 label counts (ham & spam) and 5571 messages counted
- There are 2 unique labels (ham -> good email, and spam -> bad email)
- There are 5168 unique messages and this makes sense to be less than the count (5571)
- ... because, we could have repetitions in the messages;
- ... In this case, there are 5571-5168 = 403 repeated messages
- The 'top' label (the most common label) is 'ham' with 4824 freq/repetitioins
- The 'top' message (the most common message) is "Sorry, I'll call later" with 30 freq/repetitions

Check the statistics of both labels i.e. 'spam' and 'ham' messages

In [15]:

```
messages2.groupby('label').describe()
```

Out[15]:

	message			
	count	unique	top	freq
label				
ham	4824	4515	Sorry, I'll call later	30
spam	747	653	Please call our customer service representativ...	4

Observations:

For 'ham'

- There are 4824 good emails of which 4515 are unique, which implies 309 repeated emails
- The 'top' message (the most common message) is "Sorry, I'll call later" and was repeated 30 times (i.e. frequency)

For 'spam'

- There are 747 good emails of which 653 are unique, which implies 94 repeated emails
- The 'top' message (the most common message) is "Please call our customer service representative" and was repeated 4 times (i.e. frequency)

Balance the Dataset:

- We currently have an unbalanced dataset with 4824 good email (ham) and 747 bad emails (spam), which will make out prediction skewed towards 'ham'. We seek to balance the dataset by taking equal sample sizes of both 'ham' and 'spam'.
- Currently, the total email messages is 5,574 but after making the dataset balance, we expect 747 from each category making our total to be 1,494.

In [16]:

```
# Pull out all the 'ham' and 'spam' messages separately
ham_messages = messages2.loc[messages2['label'].isin(['ham'])]
spam_messages = messages2.loc[messages2['label'].isin(['spam'])]


# Now collect a random sample of 747 for 'ham'
ham_messages = ham_messages.sample(n = 747, replace = True)

# The 'spam' messages is already 747. So no extraction here
```

In [17]:

```
# Check both messages

ham_messages
```

Out[17]:

	label	message
3574	ham	Yeah sure I'll leave in a min
111	ham	Going for dinner.msg you after.
3439	ham	awesome, how do I deal with the gate? Charles ...
4233	ham	My love ... I hope your not doing anything dra...
280	ham	You got called a tool?
...	...	...
587	ham	Pete can you please ring meive hardly gotany c...
2799	ham	I've told him that i've returned it. That shou...
3310	ham	Oh ho. Is this the first time u use these type...
1860	ham	It could work, we'll reach a consensus at the ...
974	ham	Eh u send wrongly lar...

747 rows × 2 columns

In [18]:

```
spam_messages
```

Out[18]:

	label	message
1	spam	Free entry in 2 a wkly comp to win FA Cup fina...
4	spam	FreeMsg Hey there darling it's been 3 week's n...
7	spam	WINNER!! As a valued network customer you have...
8	spam	Had your mobile 11 months or more? U R entitle...
10	spam	SIX chances to win CASH! From 100 to 20,000 po...
...	...	...
5536	spam	Want explicit SEX in 30 secs? Ring 02073162414...
5539	spam	ASKED 3MOBILE IF 0870 CHATLINES INCLU IN FREE ...
5546	spam	Had your contract mobile 11 Mnths? Latest Moto...
5565	spam	REMINDER FROM O2: To get 2.50 pounds free call...
5566	spam	This is the 2nd time we have tried 2 contact u...

747 rows × 2 columns

Join the 'ham' and 'spam' to get a full dataset

- This will be the updated "messages2"

In [19]:

```
balanced_dataset = pd.concat([ham_messages, spam_messages])

# update 'messages2'
messages2 = balanced_dataset
messages2
```

Out[19]:

	label	message
3574	ham	Yeah sure I'll leave in a min
111	ham	Going for dinner.msg you after.
3439	ham	awesome, how do I deal with the gate? Charles ...
4233	ham	My love ... I hope your not doing anything dra...
280	ham	You got called a tool?
...	...	...
5536	spam	Want explicit SEX in 30 secs? Ring 02073162414...
5539	spam	ASKED 3MOBILE IF 0870 CHATLINES INCLU IN FREE ...
5546	spam	Had your contract mobile 11 Mnths? Latest Moto...
5565	spam	REMINDER FROM O2: To get 2.50 pounds free call...
5566	spam	This is the 2nd time we have tried 2 contact u...

1494 rows × 2 columns

Shuffle the rows in the dataset

In [20]:

```
from sklearn.utils import shuffle
```

In [21]:

```
messages2 = shuffle(messages2)

messages2
```

Out[21]:

	label	message
5539	spam	ASKED 3MOBILE IF 0870 CHATLINES INCLU IN FREE ...
2287	ham	Alex knows a guy who sells mids but he's down ...
718	spam	You have WON a guaranteed £1000 cash or a £200...
776	ham	Why don't you go tell your friend you're not s...
5380	spam	You have 1 new message. Call 0207-083-6089
...	...	...
3511	ham	I'm serious. You are in the money base
1767	ham	K, want us to come by now?
5179	ham	Babe! I fucking love you too !! You know? Fuck...
2811	ham	Thinkin about someone is all good. No drugs fo...
3727	ham	Aldrine, rakhesh ex RTM here.pls call.urgent.

1494 rows × 2 columns

Re-check the statistics of both labels i.e. 'spam' and 'ham' messages

In [22]:

```
messages2.groupby('label').describe()
```

Out[22]:

	message			
	count	**unique**	**top**	**freq**
label				
ham	747	681	I cant pick the phone right now. Pls send a me...	6
spam	747	653	Please call our customer service representativ...	4

Now let us do some Feature Engineering

Definition:

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Feature engineering is fundamental to the application of machine learning and is both difficult and expensive.

(source: https://towardsdatascience.com/exploratory-data-analysis-feature-engineering-and-modelling-using-supermarket-sales-data-part-1-228140f89298 (https://towardsdatascience.com/exploratory-data-analysis-feature-engineering-and-modelling-using-supermarket-sales-data-part-1-228140f89298))

How long are the text messages?

In [23]:

```
# Create a new column on the dataset to account for email/message length


messages2['length'] = messages2['message'].apply(len)
```

```
<ipython-input-23-ce5f36531d15>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  messages2['length'] = messages2['message'].apply(len)
```

In [24]:

```
messages2
```

Out[24]:

	label	message	length
5539	spam	ASKED 3MOBILE IF 0870 CHATLINES INCLU IN FREE ...	158
2287	ham	Alex knows a guy who sells mids but he's down ...	110
718	spam	You have WON a guaranteed £1000 cash or a £200...	145
776	ham	Why don't you go tell your friend you're not s...	145
5380	spam	You have 1 new message. Call 0207-083-6089	42
...	...	...	...
3511	ham	I'm serious. You are in the money base	38
1767	ham	K, want us to come by now?	26
5179	ham	Babe! I fucking love you too !! You know? Fuck...	157
2811	ham	Thinkin about someone is all good. No drugs fo...	52
3727	ham	Aldrine, rakhesh ex RTM here.pls call.urgent.	45

1494 rows × 3 columns

Visualize the length of the messages

Check the distribution of message length

- Note that the bin size has effect on the nature of the distribution
- For more detail, you can increase the bin-size

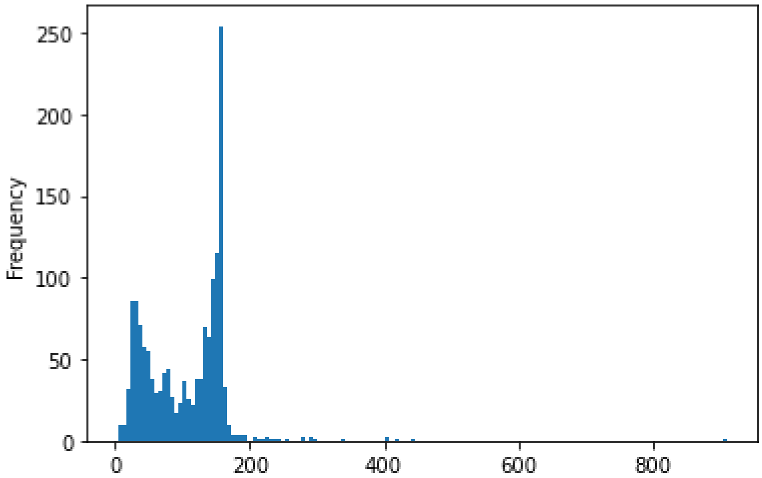
In [25]:

```
messages2['length'].plot.hist(bins = 150)

#plt.grid()
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24e8a2148b0>
```



Observations:

- We can observe that there are a few outliers, implying that there are
- ...some messages that are really long, having above 800 characters (including space)

What are the maximum and minimum email message length ?

In [26]:

```
messages2['length'].describe()
```

Out[26]:

```
count    1494.000000
mean      105.139224
std        59.720117
min         4.000000
25%        50.000000
50%       119.500000
75%       152.750000
max       910.000000
Name: length, dtype: float64
```

Observations:

- The maximum email message is 910 characters long
- The minimum email message is just 4 character long

Identify the email message of the maximum =910 character long

In [30]:

```
messages2[messages2['length'] == 910]
```

Out[30]:

	label	message	length
1084	ham	For me the love should start with attraction.i...	910

Print the entire "message" alone

In [31]:

```
messages2[messages2['length'] == 910]['message'].iloc[0]
```

Out[31]:

```
"For me the love should start with attraction.i should feel that I need he
r every time around me.she should be the first thing which comes in my tho
ughts.I would start the day and end it with her.she should be there every
time I dream.love will be then when my every breath has her name.my life s
hould happen around her.my life will be named to her.I would cry for her.w
ill give all my happiness and take all her sorrows.I will be ready to figh
t with anyone for her.I will be in love when I will be doing the craziest
things for her.love will be when I don't have to proove anyone that my gir
l is the most beautiful lady on the whole planet.I will always be singing
praises for her.love will be when I start up making chicken curry and end
up makiing sambar.life will be the most beautiful then.will get every morn
ing and thank god for the day because she is with me.I would like to say a
lot..will tell later.."
```

Identify the email message of the minimum =2 character long

In [32]:

```
messages2[messages2['length'] == 4]
```

Out[32]:

	label	message	length
2687	ham	Okie	4
3900	ham	Okie	4

Print the entire "message" alone

In [33]:

```
# we have 2 of this messages.
# We want to print all 2

total_messages = len(messages2[messages2['length'] == 4])

for i in range(total_messages):
    print(messages2[messages2['length'] == 4]['message'].iloc[i])
```

```
Okie
Okie
```

Is message length a distinguishing feature between 'ham' and 'spam'?

can we use length to determine a spam or ham email?

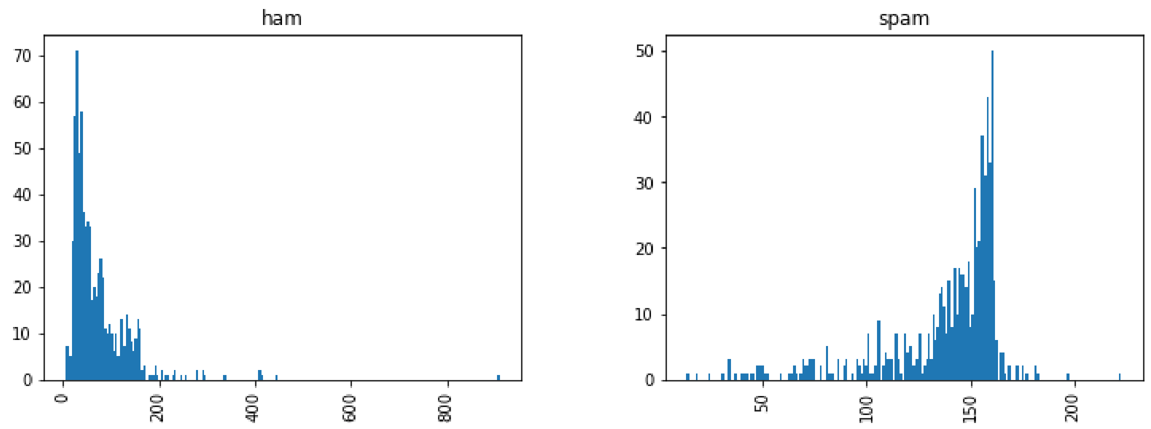
In [34]:

```
# use pandas version of facet-grid (seaborn can do this too)

messages2.hist(column  = 'length',
               by      = 'label',
               bins    =  200,
               figsize = (12,4))
```

Out[34]:

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x0000024E8A533AC0
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000024E8A53C250
>],
      dtype=object)
```



Observations:

- 'ham' emails are centered around 50 characters with a range of 0 to 200
- 'spam' emails are centered around 150 characters.

Conclusion on email message length:

- By visualization, we can tell that 'length of emails/messages' can be used
- ... as a good feature to distinguish 'ham' and 'spam' emails

- The margin is really wide. By averages: 50 - 150 (or 1:3 -ham:spam)

In [ ]:

```

```

Pre-processing the text (email messages)

- We want to convert of corpus of strings to a vector format using the Bag of words approach (i.e. each unique word in a text is represented by one number)
- So, we basically are converting the raw messages (a sequence of characters) into vectors (a sequence of numbers)
- ...to do this, we split the words in a string and store it in a list and remove stop words (like "the", "is", "and")

In [35]:

```
import string
```

In [36]:

```
# check punctuations in string-library

string.punctuation
```

Out[36]:

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

In [37]:

```
# import the stop words

from nltk.corpus import stopwords
```

How many common stopwords do we have in this library?

In [38]:

```
len(stopwords.words('english'))
```

Out[38]:

```
179
```

Check the list of these common English "stop-words"

In [39]:

```
stopwords.words('english')
```

Out[39]:

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
 "she's",
 'her',
 'hers',
 'herself',
 'it',
 "it's",
 'its',
 'itself',
 'they',
 'them',
 'their',
 'theirs',
 'themselves',
 'what',
 'which',
 'who',
 'whom',
 'this',
 'that',
 "that'll",
 'these',
 'those',
 'am',
 'is',
 'are',
 'was',
 'were',
 'be',
 'been',
 'being',
 'have',
 'has',
 'had',
 'having',
 'do',
 'does',
 'did',
```

```
 'doing',
 'a',
 'an',
 'the',
 'and',
 'but',
 'if',
 'or',
 'because',
 'as',
 'until',
 'while',
 'of',
 'at',
 'by',
 'for',
 'with',
 'about',
 'against',
 'between',
 'into',
 'through',
 'during',
 'before',
 'after',
 'above',
 'below',
 'to',
 'from',
 'up',
 'down',
 'in',
 'out',
 'on',
 'off',
 'over',
 'under',
 'again',
 'further',
 'then',
 'once',
 'here',
 'there',
 'when',
 'where',
 'why',
 'how',
 'all',
 'any',
 'both',
 'each',
 'few',
 'more',
 'most',
 'other',
 'some',
 'such',
 'no',
 'nor',
 'not',
 'only',
```

```
 'own',
 'same',
 'so',
 'than',
 'too',
 'very',
 's',
 't',
 'can',
 'will',
 'just',
 'don',
 "don't",
 'should',
 "should've",
 'now',
 'd',
 'll',
 'm',
 'o',
 're',
 've',
 'y',
 'ain',
 'aren',
 "aren't",
 'couldn',
 "couldn't",
 'didn',
 "didn't",
 'doesn',
 "doesn't",
 'hadn',
 "hadn't",
 'hasn',
 "hasn't",
 'haven',
 "haven't",
 'isn',
 "isn't",
 'ma',
 'mightn',
 "mightn't",
 'mustn',
 "mustn't",
 'needn',
 "needn't",
 'shan',
 "shan't",
 'shouldn',
 "shouldn't",
 'wasn',
 "wasn't",
 'weren',
 "weren't",
 'won',
 "won't",
 'wouldn',
 "wouldn't"]
```

In [40]:

```
# Recall original messages

messages2.head()
```

Out[40]:

	label	message	length
5539	spam	ASKED 3MOBILE IF 0870 CHATLINES INCLU IN FREE ...	158
2287	ham	Alex knows a guy who sells mids but he's down ...	110
718	spam	You have WON a guaranteed £1000 cash or a £200...	145
776	ham	Why don't you go tell your friend you're not s...	145
5380	spam	You have 1 new message. Call 0207-083-6089	42

Tokenize these mesages

- This converting a normal text string into a list of tokens (cleaned version of the words we actually want)
- This process removes the stop words and gives us a list of the words of interest
- Next, let us define a function to do this

In [41]:

```
def text_process(mess):

    """
    1. remove punctuations
    2. remmove stop words
    3. return list of clean text words

    """

    # using list comprehension
    no_punctuation = [char for char in mess if char not in string.punctuation]


    no_punctuation = ''.join(no_punctuation)

    clean_text = [word for word in no_punctuation.split() if word.lower() not in stopwo
rds.words('english')]

    return clean_text
```

In [42]:

```
messages2['message'].apply(text_process)
```

Out[42]:

```
5539    [ASKED, 3MOBILE, 0870, CHATLINES, INCLU, FREE,...
2287    [Alex, knows, guy, sells, mids, hes, south, ta...
718     [guaranteed, £1000, cash, £2000, prize, claim,...
776     [dont, go, tell, friend, youre, sure, want, li...
5380                 [1, new, message, Call, 02070836089]
                                  ...
3511                           [Im, serious, money, base]
1767                                  [K, want, us, come]
5179    [Babe, fucking, love, know, Fuck, good, hear, ...
2811                      [Thinkin, someone, good, drugs]
3727     [Aldrine, rakhesh, ex, RTM, herepls, callurgent]
Name: message, Length: 1494, dtype: object
```

In [43]:

```
# grab the first 5

messages2['message'].head(5).apply(text_process)
```

Out[43]:

```
5539    [ASKED, 3MOBILE, 0870, CHATLINES, INCLU, FREE,...
2287    [Alex, knows, guy, sells, mids, hes, south, ta...
718     [guaranteed, £1000, cash, £2000, prize, claim,...
776     [dont, go, tell, friend, youre, sure, want, li...
5380                 [1, new, message, Call, 02070836089]
Name: message, dtype: object
```

Vectorization

- convert each message into a vector that machine learning (or the SciKit Learn algorithms) models can understand

In [44]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

Let us see how this "CountVectorizer" works:

In [45]:

```
bag_of_words_transformer = CountVectorizer(analyzer = text_process).fit(messages2['mess
age'])
```

Bag of word counts for a sample email:

In [46]:

```
# create an object to store the sample message, say 'email_sample' having index 2412

email_sample = messages2['message'][2412]
email_sample
```

Out[46]:

```
"I don't know u and u don't know me. Send CHAT to 86688 now and let's find
each other! Only 150p/Msg rcvd. HG/Suite342/2Lands/Row/W1J6HL LDN. 18 year
s or over."
```

In [47]:

```
bag_of_words_3 = bag_of_words_transformer.transform([email_sample])
```

In [48]:

```
print(bag_of_words_3)
```

```
  (0, 346)	1
  (0, 373)	1
  (0, 727)	1
  (0, 1003)	1
  (0, 1388)	1
  (0, 1574)	1
  (0, 2144)	1
  (0, 3235)	2
  (0, 3413)	1
  (0, 3843)	2
  (0, 3902)	1
  (0, 4474)	1
  (0, 5095)	2
  (0, 5390)	1
```

Observation:

- This means that there are 14 unique words in sample message, after removing the stop words, three of these unique words appeared twice and we can easily confirm that using the "index" under the feature names (check next code...)

In [49]:

```
bag_of_words_transformer.get_feature_names()[3235]
```

Out[49]:

```
'dont'
```

In [51]:

```
bag_of_words_transformer.get_feature_names()[3843]
```

Out[51]:

```
'know'
```

In [52]:

```
bag_of_words_transformer.get_feature_names()[5095]
```

Out[52]:

```
'u'
```

Apply the ".transform" to the entire 'message' column in the dataframe rather one sample message (illustrated above)

In [53]:

```
# For ease:
bow_transformer = bag_of_words_transformer
```

In [54]:

```
messages2_bow = bow_transformer.transform(messages2['message'])
```

In [55]:

```
messages2_bow
```

Out[55]:

```
<1494x5478 sparse matrix of type '<class 'numpy.int64'>'
	with 18064 stored elements in Compressed Sparse Row format>
```

Observation:

- We see that the vector of the entire email message is represented as a "Sparse Matrix" (a matrix that contains more number of ZERO values than NON-ZERO values)
- We have 18064 elements that are compressed in row format. These are the number of NON-ZERO elements

Source/more on **Sparse Matrix**: [http://www.btechsmartclass.com/data_structures/sparse-matrix.html#:~:text=Sparse%20matrix%20is%20a%20matrix,only%2010%20non%2Dzero%20elements (http://www.btechsmartclass.com/data_structures/sparse-matrix.html#:~:text=Sparse%20matrix%20is%20a%20matrix,only%2010%20non%2Dzero%20elements)](http://www.btechsmartclass.com/data_structures/sparse-matrix.html#:~:text=Sparse%20matrix%20is%20a%20matrix,only%2010%20non%2Dzero%20elements).

Shape of Sparse Matrix

In [56]:

```
print(messages2_bow.shape)
```

```
(1494, 5478)
```

How many non-zero elements in the Sparse Matrix do we have?

In [57]:

```
print(messages2_bow.nnz)
```

18064

What is the Sparsity?

- This is basically comparing the number of non-zero elements (messages) to the zero elements.
- We use a formula to do this: (number of zero elements/total number of elements (m-by-n matrix))
- The density is 1 - sparsity. This compares the number of zero elements to the non-zero elements

In [58]:

```
sparsity = (messages2_bow.nnz / (messages2_bow.shape[0] * messages2_bow.shape[1]))

sparsity

print("Sparsity in percentage is {}% ".format(round(sparsity*100.0, 4)))

print("\n")

print("Density in percentage is {} % ".format(1 - round(sparsity*100.0, 4)))
```

```
Sparsity in percentage is 0.2207%


Density in percentage is 0.7793 %
```

Observation:

- With such a sparsity value, it is clear that the number of zeros in the matrix is high. This is evident in the high value of the Density (closer to 1)

Step 2 and Step 3:

- We use the TF-IDF
- **TF-IDF** means **Term Frequency- Inverse Document Frequency:** The *tf-idf* weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

- The **tf-idf weight** is composed by two(2) terms:
  - the first computes the normalized Term Frequency (TF): the number of times a word appears in a document, divided by the total number of words in that document (i.e. email)
    - **TF('word') = (Number of times term 'word' appears in a document) / (Total number of terms in the document).**
  - the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents (e.g. emails) in the corpus divided by the number of documents where the specific term appears.
    - **IDF('word') = log_e(Total number of documents / Number of documents with term 'word' in it).**

- Therefore, final value is gotten as:

```
**TF-IDF = TF('word') * IDF('word')**
```

In [59]:

```
from sklearn.feature_extraction.text import TfidfTransformer
```

==>> Let us see how this "Tfidf Transformer" works:

In [60]:

```
# For ease, let:

bow_transformer = bag_of_words_transformer
```

Apply the ".transform" to the entire 'message' column in the dataframe rather one sample message (illustrated above)

In [61]:

```
messages2_bow = bow_transformer.transform(messages2['message'])
```

In [62]:

```
tfidf_transformer = TfidfTransformer().fit(messages2_bow)
```

In [63]:

```
# Transform just one message to get the Inverse Document Frequency and
# Term Frequency (TF) relationship for email_sample (i.e. example used previously)


tfidf3 = tfidf_transformer.transform(bag_of_words_3)
```

In [64]:

```
print(tfidf3)
```

```
  (0, 5390)	0.2530024609593074
  (0, 5095)	0.2720147207017997
  (0, 4474)	0.2578774929341825
  (0, 3902)	0.2697846346883843
  (0, 3843)	0.35923064479985967
  (0, 3413)	0.21071638409659496
  (0, 3235)	0.38048187941091627
  (0, 2144)	0.21416754331854163
  (0, 1574)	0.2578774929341825
  (0, 1388)	0.2865668084174612
  (0, 1003)	0.2318594225279512
  (0, 727)	0.21995228077374937
  (0, 373)	0.1873180833175463
  (0, 346)	0.2697846346883843
```

Observation:

- The numbers are the weight values for each of these words versus the actual document
- Recall, these words can be seen using their index, as previously illustrated

In [65]:

```
# Recall the email_sample under consideration

email_sample
```

Out[65]:

```
"I don't know u and u don't know me. Send CHAT to 86688 now and let's find
each other! Only 150p/Msg rcvd. HG/Suite342/2Lands/Row/W1J6HL LDN. 18 year
s or over."
```

What is the term frequency- inverse document frequency weight of 'CHAT' in sample_email ?

In [66]:

```
# First, check which index represents 'CHAT'

bag_of_words_transformer.vocabulary_['CHAT']
```

Out[66]:

```
1003
```

Observation:

- The result vectors are arranged in descending order of the index

Result:

The term frequency-inverse document frequency weight of 'say' in email3 is gotten from

(0, 977) --> 0.23248663263329686 and is approximately **0.2325**

Now, let's explore OUTSIDE of email_sample:

What is the Inverse Document Frequency (IDF) of "house" ?

In [67]:

```
tfidf_transformer.idf_[bow_transformer.vocabulary_['house']]
```

Out[67]:

```
6.230439944144951
```

In [68]:

```
tfidf_transformer.idf_[bow_transformer.vocabulary_['pictures']]
```

Out[68]:

```
7.616734305264841
```

Train the 'spam' and 'ham' classifier

- Many other classifiers can be used but we want to use the Naive-Bayes classifier

In [69]:

```
# First, we convert the entire bag of word corpus into a tfidf corpus at once.
# This similar to what we did with bow_of_words_3 (for email_sample)



tfidf_all_messages = tfidf_transformer.transform(messages2_bow)
```

==>> Let us see how this "MultinomialNB" works:

In [70]:

```
spam_detect_model = MultinomialNB().fit(X = tfidf_all_messages,
                                        y = messages2['label'])
```

Classify a single random message to see how the prediction will do:

In [71]:

```
spam_detect_model.predict(tfidf3)
```

Out[71]:

```
array(['spam'], dtype='<U4')
```

In [72]:

```
spam_detect_model.predict(tfidf3)[0]
```

Out[72]:

```
'spam'
```

Result:

- The model detects that the 'email_sample' message will be 'ham' (good message)

- Check if the above prediction is true:

In [73]:

```
messages2['label'][2412]  # from the original
```

Out[73]:

```
'spam'
```

Result:

- Yes, our model seems to be predicting correctly.

Now, let's do this prediction for all the emails in the dataset

In [74]:

```
predict_all = spam_detect_model.predict(tfidf_all_messages)
```

In [75]:

```
predict_all
```

Out[75]:

```
array(['spam', 'ham', 'spam', ..., 'ham', 'ham', 'ham'], dtype='<U4')
```

Split data to Training and Testing data

In [76]:

```
X_train, X_test, y_train, y_test = train_test_split(messages2['message'],
                                                    messages2['label'],
                                                    test_size    = 0.20,    # using 20%
 as test_data
                                                    random_state = 42)
# Notes:
# X represents 'message'
# y represents 'label'
```

Notes:

- Now we would use "Pipeline" rather doing all the long previous steps we went through
- To do this:
  - Pass in a list of *everything* you want to do into the 'Pipeline'
  - *everything* here would mean a tuple having ('name of step', 'calculation/function of that step')

In [77]:

```
pipeline = Pipeline([
    ('bag of words', CountVectorizer(analyzer = text_process)),
    ('tfidf', TfidfTransformer()),
    ('Train on Model', MultinomialNB())  # you can change this 'Classifier' e.g. to Ran
domForestClassfier etc
])
```

In [78]:

```
model = pipeline.fit(X_train, y_train)
model
```

Out[78]:

```
Pipeline(steps=[('bag of words',
                 CountVectorizer(analyzer=<function text_process at 0x0000
024E8AAF7820>)),
                ('tfidf', TfidfTransformer()),
                ('Train on Model', MultinomialNB())])
```

Do the Prediction of the Test Data

In [79]:

```
predictions = model.predict(X_test)
```

In [80]:

```
predictions
```

Out[80]:

```
array(['ham', 'spam', 'ham', 'ham', 'spam', 'spam', 'spam', 'ham', 'spam',
       'ham', 'spam', 'ham', 'spam', 'spam', 'ham', 'spam', 'spam', 'ham',
       'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'spam', 'spam', 'spam',
       'ham', 'ham', 'ham', 'ham', 'spam', 'ham', 'ham', 'ham', 'ham',
       'ham', 'ham', 'ham', 'spam', 'ham', 'spam', 'ham', 'ham', 'ham',
       'ham', 'ham', 'ham', 'spam', 'ham', 'spam', 'ham', 'ham', 'ham',
       'spam', 'spam', 'spam', 'spam', 'ham', 'ham', 'ham', 'spam',
       'spam', 'spam', 'ham', 'spam', 'ham', 'ham', 'spam', 'ham', 'spam',
       'ham', 'ham', 'ham', 'ham', 'spam', 'spam', 'spam', 'spam', 'ham',
       'ham', 'spam', 'spam', 'ham', 'ham', 'spam', 'ham', 'spam', 'spam',
       'spam', 'spam', 'spam', 'ham', 'spam', 'spam', 'spam', 'ham',
       'spam', 'ham', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam',
       'spam', 'ham', 'spam', 'ham', 'ham', 'ham', 'spam', 'ham', 'ham',
       'spam', 'spam', 'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'spam',
       'ham', 'spam', 'ham', 'spam', 'spam', 'spam', 'ham', 'ham', 'ham',
       'ham', 'ham', 'spam', 'ham', 'spam', 'ham', 'ham', 'spam', 'ham',
       'ham', 'spam', 'ham', 'spam', 'spam', 'ham', 'ham', 'ham', 'ham',
       'spam', 'ham', 'spam', 'spam', 'spam', 'ham', 'ham', 'spam', 'ham',
       'spam', 'ham', 'spam', 'spam', 'spam', 'ham', 'ham', 'ham', 'ham',
       'ham', 'ham', 'spam', 'spam', 'spam', 'spam', 'spam', 'ham', 'ham',
       'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'spam', 'ham', 'ham',
       'spam', 'spam', 'ham', 'ham', 'ham', 'spam', 'spam', 'spam', 'ham',
       'ham', 'spam', 'ham', 'ham', 'spam', 'spam', 'ham', 'ham', 'ham',
       'ham', 'ham', 'ham', 'spam', 'ham', 'ham', 'ham', 'spam', 'ham',
       'ham', 'spam', 'spam', 'ham', 'ham', 'spam', 'spam', 'spam', 'ham',
       'spam', 'ham', 'ham', 'spam', 'spam', 'ham', 'spam', 'spam', 'ham',
       'spam', 'ham', 'spam', 'spam', 'ham', 'spam', 'ham', 'spam',
       'spam', 'ham', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam',
       'spam', 'ham', 'spam', 'spam', 'ham', 'spam', 'ham', 'ham', 'ham',
       'spam', 'ham', 'ham', 'spam', 'spam', 'ham', 'spam', 'spam',
       'spam', 'ham', 'spam', 'spam', 'spam', 'spam', 'spam', 'ham',
       'spam', 'ham', 'ham', 'spam', 'spam', 'ham', 'spam', 'ham', 'spam',
       'spam', 'spam', 'ham', 'ham', 'spam', 'spam', 'spam', 'spam',
       'spam', 'spam', 'spam', 'ham', 'spam', 'ham', 'ham', 'ham', 'spam',
       'spam'], dtype='<U4')
```

Confusion Matrix

```
                                              Predicted
                          ------------------------------------------------------------------
----
                         |        ham                          |         spam
       |--------------------------------------------------------------------------
-----
Actual |ham      |  True ham  (True Negative)        |    False spam (False Positi
ve)
       |spam     |  False ham (False Negative)       |    True spam  (True Positiv
e)
```

In [81]:

```
print(confusion_matrix(y_true = y_test,
                       y_pred = predictions))
```

```
[[136   8]
 [ 12 143]]
```

Classification report

In [82]:

```
print(classification_report(y_true = y_test,
                            y_pred = predictions))
```

```
              precision    recall  f1-score   support

         ham       0.92      0.94      0.93       144
        spam       0.95      0.92      0.93       155

    accuracy                           0.93       299
   macro avg       0.93      0.93      0.93       299
weighted avg       0.93      0.93      0.93       299
```

Results:

- Generally, the accuracy of the model seems very good being 93% accurate. But, our focus will not be on *Accuracy*, rather will be on *Precision*.
- **Precision:** is a good measure to determine, when we prioritize the costs of False-Positive. If we take Positive to mean 'spam' and Negative to mean 'ham'; then a false positive would mean that a ham or good email (actual negative) has been identified as spam or bad email (predicted spam). ***The result of this becomes that the email user might lose important emails if the precision is not high for the model.***
- ham:
  - The *Precision* is 92% which tells very well on our model in classifying good emails.
- spam:
  - The *Precision* is 95% which tells us that our model is doing fine in detecting spam emails, although not as 'ham'

Assumming we wanted to predict new email

In [83]:

```
new_email = ['I am so grateful and excited. I got my the Job! Hurray!!!!!!']

pipeline.predict(new_email)
```

Out[83]:

```
array(['ham'], dtype='<U4')
```

In [84]:

```
new_email2 = ['We will give you $1,000 for sending an e-mail to your friends.  AB Maili
ng, Inc. is proud to anounce the start of a new contest.  Each day untilJanuary, 31 199
9, one lucky Internet or AOL user who forwards our advertisement to their friends will
 be randomly picked to receive $1,000!  You could be the winner!']

pipeline.predict(new_email2)




# This example is from MIT: http://web.mit.edu/network/spam/examples/
# Specifically example "Jan 1999"
```

Out[84]:

```
array(['spam'], dtype='<U4')
```

Note:

- We can change the 'Train on Model -Classifier' (currently Naive Bayes) to other types of classifiers or ensemble methods
  - Other Classifiers:
    - Decision Tree,
    - K-Nearest Neigbor
    - Artificial Neural Networks
    - Logistic Regression
    - Support Vector Machine etc
    - Ensemble methods:

```
* Random Forest
* Bagging etc
```

Using the *Logistic Regression* classifier:

In [85]:

```
from sklearn.linear_model import LogisticRegression
```

In [86]:

```
pipeline = Pipeline([
    ('bag of words', CountVectorizer(analyzer = text_process)),
    ('tfidf', TfidfTransformer()),
    ('Train on Model', LogisticRegression())
])
```

In [87]:

```
pipeline.fit(X_train, y_train)
```

Out[87]:

```
Pipeline(steps=[('bag of words',
                 CountVectorizer(analyzer=<function text_process at 0x0000
024E8AAF7820>)),
                ('tfidf', TfidfTransformer()),
                ('Train on Model', LogisticRegression())])
```

In [88]:

```
predictions = pipeline.predict(X_test)
```

In [89]:

```
print(classification_report(y_true = y_test,
                            y_pred = predictions))
```

```
              precision    recall  f1-score   support

         ham       0.91      0.96      0.93       144
        spam       0.96      0.91      0.93       155

    accuracy                           0.93       299
   macro avg       0.93      0.93      0.93       299
weighted avg       0.93      0.93      0.93       299
```

Using an Ensemble method of classifier: *Random Forest* classifier:

In [90]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [91]:

```
pipeline = Pipeline([
    ('bag of words', CountVectorizer(analyzer = text_process)),
    ('tfidf', TfidfTransformer()),
    ('Train on Model', RandomForestClassifier())
])
```

In [92]:

```
pipeline.fit(X_train, y_train)
```

Out[92]:

```
Pipeline(steps=[('bag of words',
                 CountVectorizer(analyzer=<function text_process at 0x0000
024E8AAF7820>)),
                ('tfidf', TfidfTransformer()),
                ('Train on Model', RandomForestClassifier())])
```

In [93]:

```
predictions = pipeline.predict(X_test)
```

In [94]:

```
print(classification_report(y_true = y_test,
                            y_pred = predictions))
```

```
              precision    recall  f1-score   support

         ham       0.89      1.00      0.94       144
        spam       1.00      0.88      0.94       155

    accuracy                           0.94       299
   macro avg       0.94      0.94      0.94       299
weighted avg       0.95      0.94      0.94       299
```

Concluding Thoughts:

- Some Data Scientist prefer to use an ensembler method for classification often because *it is a set of classifiers whose individual decisions are combined in some way to classify new examples*.
- From our few trials of both *single classifiers* and *set of classifiers* (ensembler method) does well. I will give preference to the Logistic Regression Classifier, simply because it balances giving a higher prediction for 'spam' and not lowering the prediction for 'ham' so much. Recall our attention is still on *Precision*.
- I did not choose RandomForestClassifier as my preferred because, although the 'spam'-detection precision is very high or seems perfect, I think, it lowered so much of the 'ham' precision. Again, this is just my preference.
- As a future step, I will want to try other classifiers. In my opinion, choosing the classifier would depend mostly on the Data Scientist or Team and what they want to achieve.