EMAIL SPAM DETECTION WITH MACHINE LEARNING

Weve all been the recipient of spam emails before. Spam mail, or junk mail, is a type of email that is sent to a massive number of users at one time, frequently containing cryptic messages, scams, or most dangerously, phishing content

In [27]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import string
import pprint
from collections import Counter
from sklearn import feature_extraction, model_selection, naive_bayes, metrics, svm
from IPython.display import Image
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

In [6]:
```
# Importing data
df = pd.read_csv('C:/Users/Gayatri/Downloads/spam.csv', encoding='latin-1')
df.head(10)
```

Out[6]:

	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
5	spam	FreeMsg Hey there darling it's been 3 week's n...	NaN	NaN	NaN
6	ham	Even my brother is not like to speak with me. ...	NaN	NaN	NaN
7	ham	As per your request 'Melle Melle (Oru Minnamin...	NaN	NaN	NaN
8	spam	WINNER!! As a valued network customer you have...	NaN	NaN	NaN
9	spam	Had your mobile 11 months or more? U R entitle...	NaN	NaN	NaN

In [8]:
```
# EDA
df = df.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
df = df.rename(columns={"v1":"label", "v2":"sms"})
```

In [9]:
```
df
```

Out[9]:

	label	sms
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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

In [10]:
```
#Number of observations in each label spam and ham
df.label.value_counts()
```

Out[10]:
```
ham     4825
spam     747
Name: label, dtype: int64
```

In [11]:
```
df.describe()
```

Out[11]:

	label	sms
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

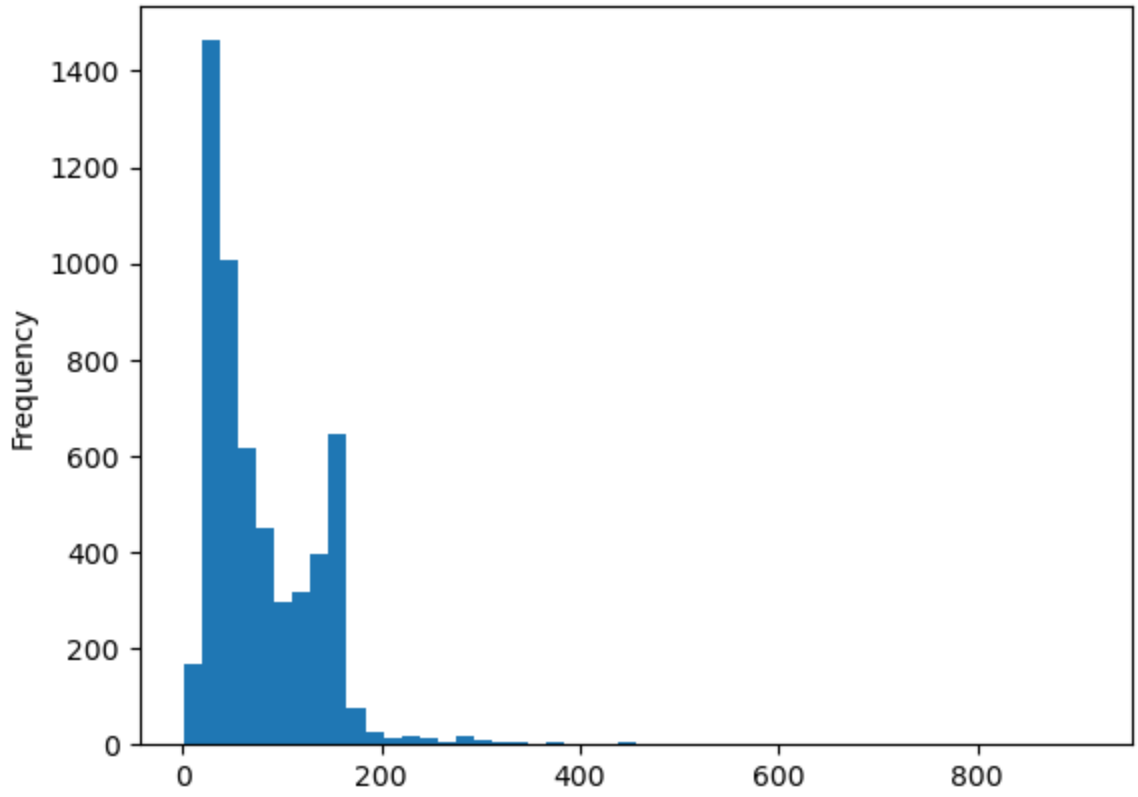
In [12]:
```
df['length'] = df['sms'].apply(len)
df.head(5)
```

Out[12]:

	label	sms	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

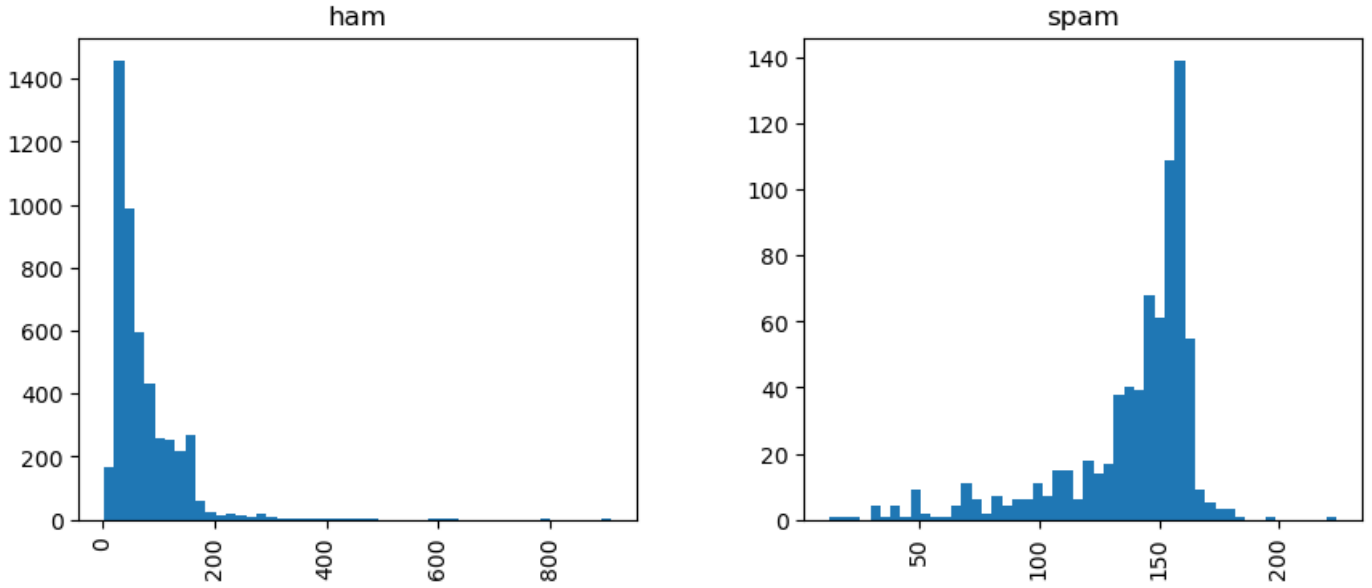
In [15]:
```
#Visualization
df['length'].plot(bins=50, kind='hist')
```

Out[15]:
```
<AxesSubplot:ylabel='Frequency'>
```

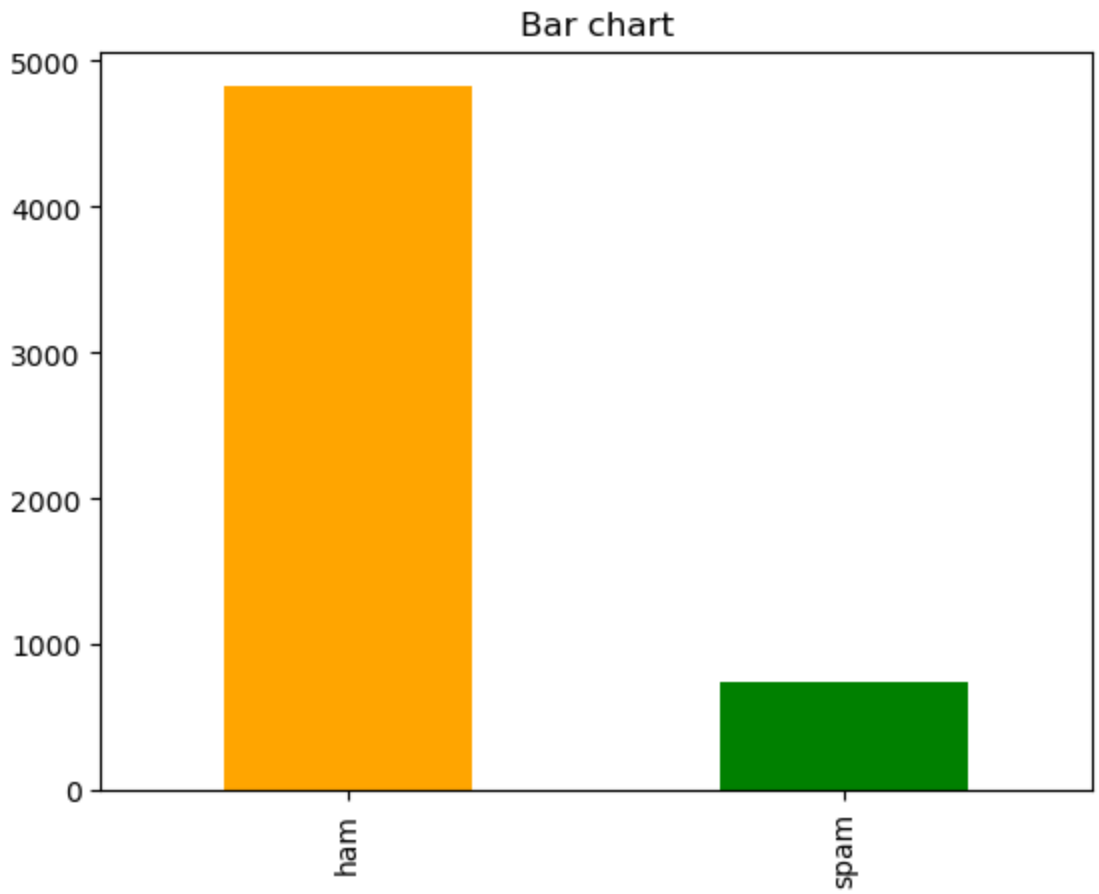


In [16]:
```
df.hist(column='length', by='label', bins=50,figsize=(10,4))
```

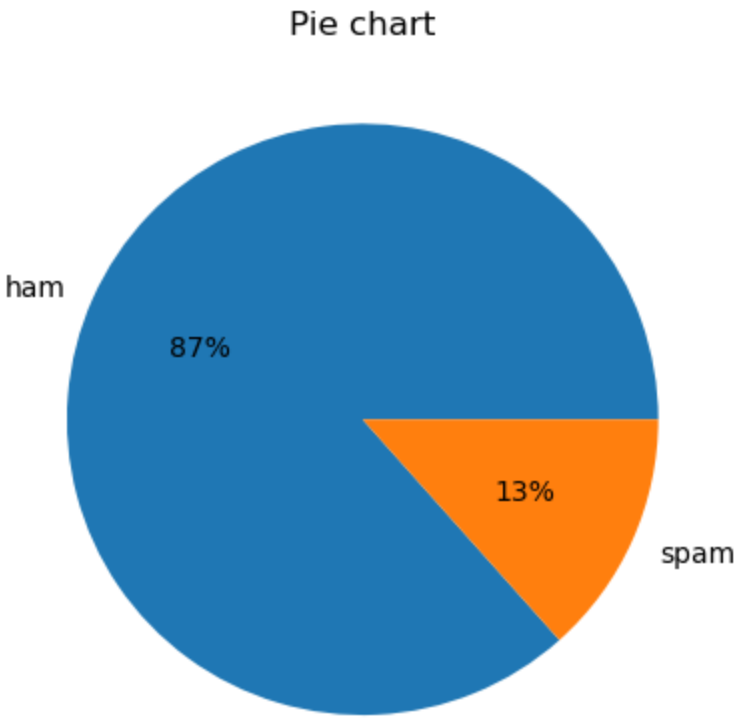
Out[16]:
```
array([<AxesSubplot:title={'center':'ham'}>,
       <AxesSubplot:title={'center':'spam'}>], dtype=object)
```



In [17]:
```
count_Class=pd.value_counts(df["label"], sort= True)
count_Class.plot(kind= 'bar', color= ["orange", "green"])
plt.title('Bar chart')
plt.show()
```



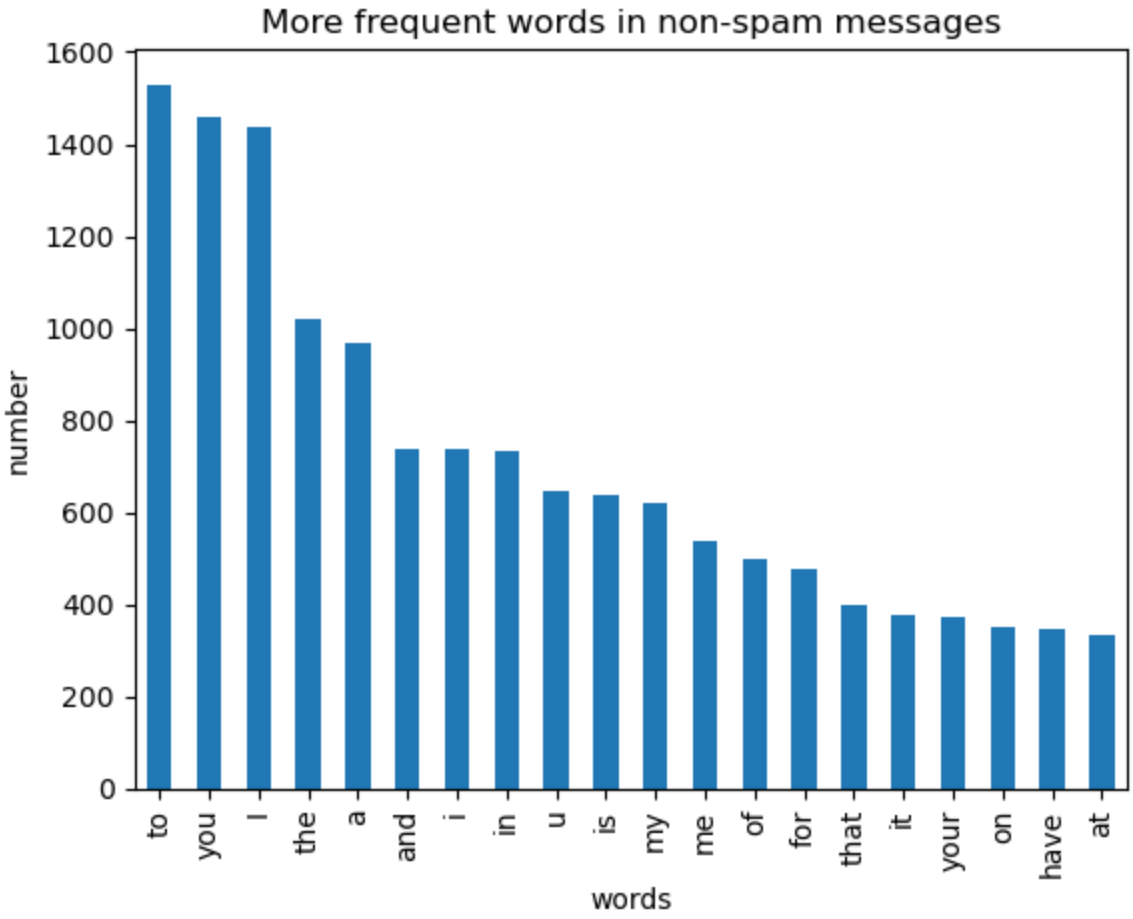
In [18]:
```
count_Class.plot(kind = 'pie',  autopct='%1.0f%%')
plt.title('Pie chart')
plt.ylabel('')
plt.show()
```



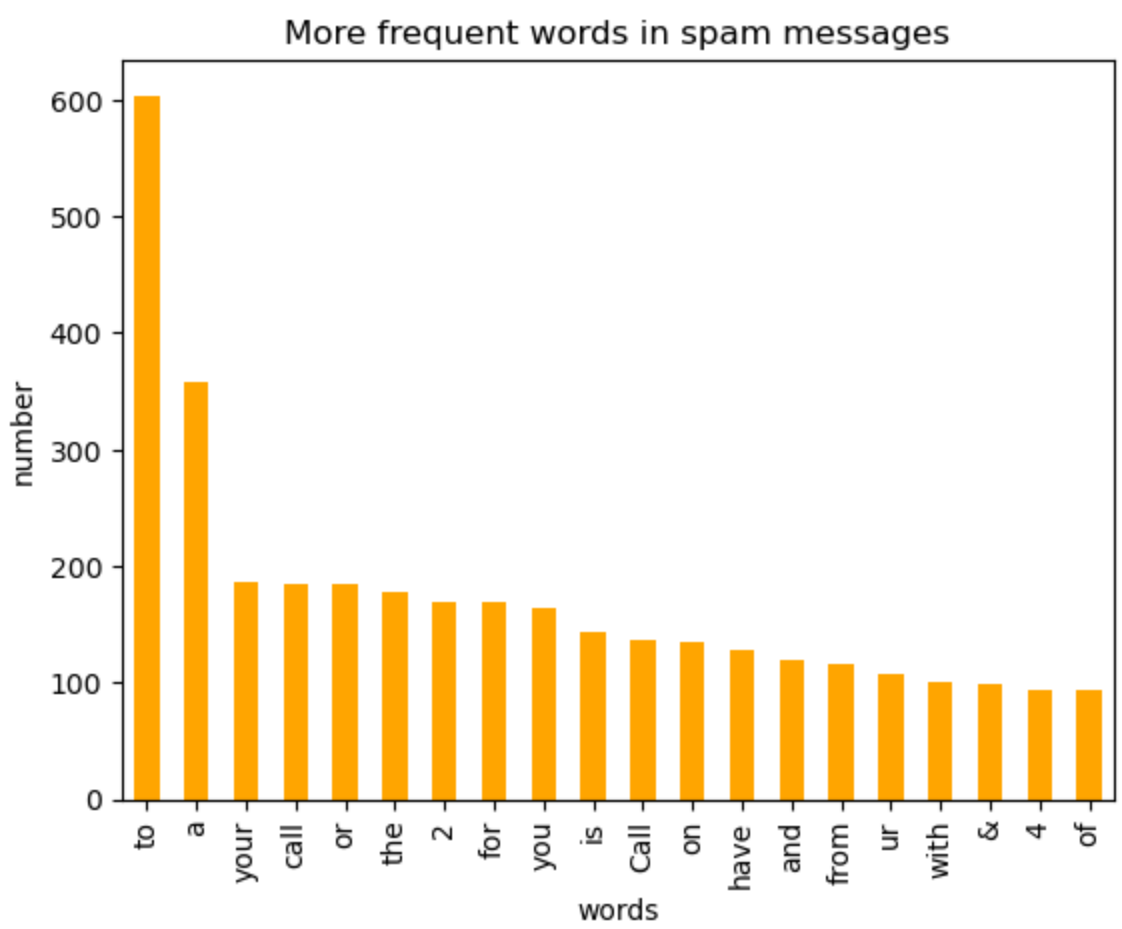
In [19]:
```
count1 = Counter(" ".join(df[df['label']=='ham']["sms"]).split()).most_common(20)
df1 = pd.DataFrame.from_dict(count1)
df1 = df1.rename(columns={0: "words in non-spam", 1 : "count"})
count2 = Counter(" ".join(df[df['label']=='spam']["sms"]).split()).most_common(20)
```

```
df2 = pd.DataFrame.from_dict(count2)
df2 = df2.rename(columns={0: "words in spam", 1 : "count_"})
```

In [20]:
```
df1.plot.bar(legend = False)
y_pos = np.arange(len(df1["words in non-spam"]))
plt.xticks(y_pos, df1["words in non-spam"])
plt.title('More frequent words in non-spam messages')
plt.xlabel('words')
plt.ylabel('number')
plt.show()
```



In [21]:
```
df2.plot.bar(legend = False, color = 'orange')
y_pos = np.arange(len(df2["words in spam"]))
plt.xticks(y_pos, df2["words in spam"])
plt.title('More frequent words in spam messages')
plt.xlabel('words')
plt.ylabel('number')
plt.show()
```



In [22]:
```
df.loc[:,'label'] = df.label.map({'ham':0, 'spam':1})
print(df.shape)
df.head()
```

```
(5572, 3)
```

Out[22]:

	label	sms	length
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

Words Approach

What we have here in our data set is a large collection of text data (5,572 rows of data). Most ML algorithms rely on numerical data to be fed into them as input, and email/sms messages are usually text heavy. We need a way to represent text data for machine learning algorithm and the bag-of-words model helps us to achieve that task. It is a way of extracting features from the text for use in machine learning algorithms. In this approach, we use the tokenized words for each observation and find out the frequency of each token. Using a process which we will go through now, we can convert a collection of documents to a matrix, with each document being a row and each word(token) being the column, and the corresponding (row,column) values being the frequency of occurrence of each word or token in that document.

For example:

Lets say we have 4 documents as follows:

['Hello, how are you!', 'Win money, win from home.', 'Call me now', 'Hello, Call you tomorrow?']

Our objective here is to convert this set of text to a frequency distribution matrix.

Implementation of Bag of Words Approach

Step 1: Convert all strings to their lower case form

In [23]:
```
documents = ['Hello, how are you!',
             'Win money, win from home.',
             'Call me now.',
             'Hello, Call hello you tomorrow?']

lower_case_documents = []
lower_case_documents = [d.lower() for d in documents]
print(lower_case_documents)
```

```
['hello, how are you!', 'win money, win from home.', 'call me now.', 'hello, call hello
you tomorrow?']
```

In [25]:
```
# Step 2: Removing all punctuations
sans_punctuation_documents = []


for i in lower_case_documents:
    sans_punctuation_documents.append(i.translate(str.maketrans("","", string.punctuatio

sans_punctuation_documents
```

Out[25]:
```
['hello how are you',
 'win money win from home',
 'call me now',
 'hello call hello you tomorrow']
```

In [26]:
```
# Step 3: Tokenization
preprocessed_documents = [[w for w in d.split()] for d in sans_punctuation_documents]
preprocessed_documents
```

Out[26]:
```
[['hello', 'how', 'are', 'you'],
 ['win', 'money', 'win', 'from', 'home'],
 ['call', 'me', 'now'],
 ['hello', 'call', 'hello', 'you', 'tomorrow']]
```

In [28]:
```
# Step 4: Count frequencies
frequency_list = []
frequency_list = [Counter(d) for d in preprocessed_documents]
pprint.pprint(frequency_list)
```

```
[Counter({'hello': 1, 'how': 1, 'are': 1, 'you': 1}),
 Counter({'win': 2, 'money': 1, 'from': 1, 'home': 1}),
 Counter({'call': 1, 'me': 1, 'now': 1}),
 Counter({'hello': 2, 'call': 1, 'you': 1, 'tomorrow': 1})]
```

In [29]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_vector = CountVectorizer()
```

Data preprocessing with CountVectorizer()

In above step, we implemented a version of the CountVectorizer() method from scratch that entailed cleaning our data first. This cleaning involved converting all of our data to lower case and removing all punctuation marks. CountVectorizer() has certain parameters which take care of these steps for us. They are:

lowercase = True

The lowercase parameter has a default value of True which converts all of our text to its lower case form.

token_pattern = (?u)\b\w\w+\b

The token_pattern parameter has a default regular expression value of (?u)\b\w\w+\b which ignores all punctuation marks and treats them as delimiters, while accepting alphanumeric strings of length greater than or equal to 2, as individual tokens or words.

stop_words

The stop_words parameter, if set to english will remove all words from our document set that match a list of English stop words which is defined in scikit-learn. Considering the size of our dataset and the fact that we are dealing with SMS messages and not larger text sources like e-mail, we will not be setting this parameter value.

In [30]:
```
count_vector.fit(documents)
count_vector.get_feature_names()
```

Out[30]:
```
['are',
 'call',
 'from',
 'hello',
 'home',
 'how',
 'me',
 'money',
 'now',
 'tomorrow',
 'win',
 'you']
```

In [31]:
```
doc_array = count_vector.transform(documents).toarray()
doc_array
```

Out[31]:
```
array([[1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 2, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
       [0, 1, 0, 2, 0, 0, 0, 0, 0, 1, 0, 1]], dtype=int64)
```

In [32]:
```
frequency_matrix = pd.DataFrame(doc_array, columns = count_vector.get_feature_names())
frequency_matrix
```

Out[32]:

	are	call	from	hello	home	how	me	money	now	tomorrow	win	you
0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	0	1	0	1	0	0	1	0	0	2	0
2	0	1	0	0	0	0	1	0	1	0	0	0
3	0	1	0	2	0	0	0	0	0	1	0	1

Model Building

In [33]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df['sms'],
                                                    df['label'],test_size=0.20,
                                                    random_state=1)
```

In [34]:
```
# Fit the training data and then return the matrix
training_data = count_vector.fit_transform(X_train)

# Transform testing data and return the matrix.
testing_data = count_vector.transform(X_test)
```

In [35]:
```
from sklearn.naive_bayes import MultinomialNB
naive_bayes = MultinomialNB()
naive_bayes.fit(training_data,y_train)
```

Out[35]:
```
MultinomialNB()
```

In [36]:
```
predictions = naive_bayes.predict(testing_data)
```

In [37]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('Accuracy score: {}'.format(accuracy_score(y_test, predictions)))
print('Precision score: {}'.format(precision_score(y_test, predictions)))
print('Recall score: {}'.format(recall_score(y_test, predictions)))
print('F1 score: {}'.format(f1_score(y_test, predictions)))
```

```
Accuracy score: 0.9847533632286996
Precision score: 0.9420289855072463
Recall score: 0.935251798561151
F1 score: 0.9386281588447652
```

In [ ]: