In [5]:
```
import pandas as pd
data=pd.read_csv('train.tsv',sep='\t')
```

In [6]:
```
data.describe()
```

Out[6]:

	PhraseId	SentenceId	Sentiment
count	156060.000000	156060.000000	156060.000000
mean	78030.500000	4079.732744	2.063578
std	45050.785842	2502.764394	0.893832
min	1.000000	1.000000	0.000000
25%	39015.750000	1861.750000	2.000000
50%	78030.500000	4017.000000	2.000000
75%	117045.250000	6244.000000	3.000000
max	156060.000000	8544.000000	4.000000

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 156060 entries, 0 to 156059
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype 
---  ------      --------------   ----- 
 0   PhraseId    156060 non-null  int64 
 1   SentenceId  156060 non-null  int64 
 2   Phrase      156060 non-null  object
 3   Sentiment   156060 non-null  int64 
dtypes: int64(3), object(1)
memory usage: 4.8+ MB
```

In [9]:
```
#sentiment value counts
data.Sentiment.value_counts()
```

Out[9]:
```
2    79582
3    32927
1    27273
4     9206
0     7072
Name: Sentiment, dtype: int64
```

To proceed further with the sentiment analysis we need to do text classification. We can use 'bag of words (BOW)' model for the analysis. In laymen terms BOW model converts text in the form of numbers which can then be used in an algorithm for analysis.

Specifically BOW model is used for feature extraction in text data. It returns a vector with all the words and number of times each word is repeated. It is known as BOW because it is only concerned with the number of times a word is repeated rather than order of words. Let's take an example to understand it better (assume each document contains a sentence only):

Doc1: Switzerland is a beautiful country. Doc2: India is a country of smart IT professionals. Doc3: USA is a country of opportunities.

More the content in each document lengthier would be the length of each vector (will contain lot of zeros). Basically doc vectors would be a sparse vectors if documents are too large. Sparse vectors need lot of memory for storage and due to length even computation becomes slow. In order to reduce the length of the sparse vectors one may use the technique like stemming, lematization, converting to lower case or ignoring stop-words e.t.c.

Now, we will generate DTM using CountVectorizer module of scikit-learn. To read more about the arguments of CountVectorizer you may visit here. As discussed above we will use:

tokenizer = Overrides the string tokenization step, we generatre tokenizer from NLTK's Regex tokenizer (by default: None) lowercase = True (no need to use, as it is set True by default) stop_words = 'english' (by default None is used, to improve the result we can provide custom made list of stop words) ngram_range = (1,1) (by defualt its (1,1) i.e strictly monograms will be used, (2,2) only bigrams while (1,2) uses both)

In [11]:
```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import RegexpTokenizer
token = RegexpTokenizer(r'[a-zA-Z0-9]+')
cv = CountVectorizer(stop_words='english',ngram_range = (1,1),tokenizer = token.t
text_counts = cv.fit_transform(data['Phrase'])
```

In [13]:
```
#splitting data into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(text_counts, data['Sentiment'
```

Now we have the training and testing data. We should start the analysis. Our analysis (as most of ML analysis) will be in 5 steps(a mneumonic to remember them is DC-FEM remember as DC Female or District of Columbia Fire and Emergency Medical service):

1. Defining the model
2. Compiling the model
3. Fitting the model
4. Evaluating the model
5. Making predictions with the model

In [14]:
```
#we use the sklearn naive bayes model
from sklearn.naive_bayes import MultinomialNB

MNB = MultinomialNB()
MNB.fit(X_train, Y_train)
```

Out[14]:
```
MultinomialNB()
```

In [16]:
```
#evaluating the nb model

from sklearn import metrics
predicted = MNB.predict(X_test)
accuracy_score = metrics.accuracy_score(predicted, Y_test)


print(str('{:04.2f}'.format(accuracy_score*100))+'%')
```

```
60.25%
```

Trying different n grams

In [20]:
```
#from sklearn.feature_extraction import CountVectorizer
#from nltk.tokenize import RegexpTokenizer
#token = RegexpTokenizer(r'[A-Za-z0-9]+')
cv = CountVectorizer(stop_words='english', ngram_range = (2,2), tokenizer = toker
text_counts = cv.fit_transform(data['Phrase'])

#from sklearn.model_selection import train_test_split()
X_train, X_test, Y_train, Y_test = train_test_split(text_counts, data['Sentiment'

#Defining the model-> we will use MultinomialNB

#Compiling the model -> We will import precompiled MNB from sklearn library
#from sklearn.naive_bayes import MultinomialNB

#Fitting the model
MNB = MultinomialNB()
MNB.fit(X_train, Y_train)

#Evaulating the model
#form sklearn import metrics
accuracy_score = metrics.accuracy_score(MNB.predict(X_test), Y_test)
print(str('{:04.2f}'.format(accuracy_score*100))+'%')
```

```
60.42%
```

TF-IDF: Term Frequency-Inverse Document Frequency Let's use TF-IDF here product of term frequency and inverse document frequency is used. Term frequency is how frequently a terms has appeared in a document. Let's say a term appears f times in a document with d words. Term Frequency = f/d IDF is inverse document frequency. If a corpus contains N documents and the term of our interest appears only in D documents then IDF is: IDF = log(N/D) TF-IDF is product of Term Frequncy and Inverse Document Frequency. TF-IDF shows the rarity of a word in the corpus. If a word is rare then probably its a signature word for a particular sentiment/information.

In [30]:
```
#trying different non bayesian algorithms
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import TfidfVectorizer

X_train, X_test, Y_train, Y_test = train_test_split(text_counts, data['Sentiment'

tfidf = TfidfVectorizer()
text_count_2 = tfidf.fit_transform(data['Phrase'])

#splitting the data in test and training
#from sklearn.model_selection() import train_test_split()
x_train, x_test, y_train, y_test = train_test_split(text_count_2, data['Sentiment

SGDC = SGDClassifier()
LSVC = LinearSVC()

#on TF-IDF data
LSVC.fit(x_train, y_train)
accuracy_score_lsvc = metrics.accuracy_score(LSVC.predict(x_test), y_test)
print('accuracy_score_lsvc = '+str('{:4.2f}'.format(accuracy_score_lsvc*100))+'%'

SGDC.fit(x_train, y_train)
accuracy_score_sgdc = metrics.accuracy_score(SGDC.predict(x_test), y_test)
print('accuracy_score_sgdc = '+str('{:4.2f}'.format(accuracy_score_sgdc*100))+'%'

#on CountVectorize data
LSVC.fit(X_train, Y_train)
accuracy_score_lsvc_CV = metrics.accuracy_score(LSVC.predict(X_test), Y_test)
print('accuracy_score_lsvc_cv = '+str('{:4.2f}'.format(accuracy_score_lsvc_CV*100

SGDC.fit(X_train, Y_train)
accuracy_score_sgdc_CV = metrics.accuracy_score(SGDC.predict(X_test), Y_test)
print('accuracy_score_sgdc_cv = '+str('{:4.2f}'.format(accuracy_score_sgdc_CV*100
```

```
accuracy_score_lsvc = 63.89%
accuracy_score_sgdc = 56.48%
accuracy_score_lsvc_cv = 62.67%
accuracy_score_sgdc_cv = 56.56%
```

In [31]:
```
print('this notebook was created by aarya jha for ai,ml honors course expt 6')
```

```
this notebook was created by aarya jha for ai,ml honors course expt 6
```

In [ ]:
```

```