Thomas Nowakowski Project 9 - Natural Language Processing - US Airline Sentiment

In [15]:
```
# Importing Libraries

from bs4 import BeautifulSoup # for the removal of HTML tags

import nltk

from nltk.tokenize.toktok import ToktokTokenizer # import tokenizer for further feeding into
                                              # steemer/leminizer since these take tokens not strings
from nltk import word_tokenize  # another tokenizer to have some hyperparameter options

from nltk.tokenize import TweetTokenizer # this is the one that worked - others did not

import warnings
warnings.filterwarnings('ignore')

# from nltk.corpus import stopwords, wordnet
from nltk.corpus import stopwords, wordnet # import stopwords to apply on text

from nltk.stem import LancasterStemmer, WordNetLemmatizer  # different Lemminizers to try
                                                           # as hyperparameteres
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.stem.wordnet import WordNetLemmatizer


# import spacy # import a non NLTK lemminzer, we will test accuracy results with stemming and lemminizer as
             # part of my hyper parameter adjustment

import unicodedata  # to remove any accented characters that are often found in US based English
                    # because of the Spanish influence
import contractions

import re # pythons special character and punctuation mark removal

import pandas as pd # pandas dataframes
import numpy as np

import contractions
```

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

In [16]:
```
pwd
```
Out[16]: 'C:\\Users\\Tomek\\Desktop\\Artificial Intelligence\\PROJECTS\\Project 9 -NLP'

In [17]:
```
# import the  dataset
data=pd.read_csv('tweets.csv')
```

In [18]:
```
# print data shape
data.shape
```
Out[18]: (14640, 15)

In [19]:
```
# data describe ?????
# data.describe()
```

Understanding of data columns

In [20]:
```
#  drop all columns except for  "text" and " airline sentiment"
data1=data.loc[:14640,['text','airline_sentiment']]
```

In [21]:
```
# THE OTHER WAYTO GET THE DATASET WE WANT USING DROP
#  drop all columns except for  "text" and " airline sentiment"
# define data1 as the varable with all the excess columns dropped
# data1=data.drop(['tweet_id','airline_sentiment_confidence','negativereason','negativereason_confidence','airline','airline_sentiment_gold','name','negativereason_gold','retweet_count','tweet_coord','tweet_created','tweet_location','user_timezone'],axis=1)
```

In [22]:
```
# Check the data shape
data1.shape
```
Out[22]: (14640, 2)

In [23]:
```
# print first 5 rows of data
data1.head()
```
Out[23]:

	text	airline_sentiment
0	@VirginAmerica What @dhepburn said.	neutral
1	@VirginAmerica plus you've added commercials t...	positive
2	@VirginAmerica I didn't today... Must mean I n...	neutral
3	@VirginAmerica it's really aggressive to blast...	negative
4	@VirginAmerica and it's a really big bad thing...	negative

In [ ]:

Text Pre-processing : Data Preparation

In [24]:
```
# since there are a reasonable number of records, 14,640 there is no need to  limit
# the amount of data for processing, we will use the full set available
```

In [25]:
```
# A)  HTML Tag removal

# BeautifulSoup library already imported
def  strip_html (text):
     soup=BeautifulSoup(text,'html.parser')
     clean_text=soup.get_text()
     return clean_text

data1['text']=data1['text'].apply(lambda x:strip_html(x))

data1.head()
```

Out[25]:

	text	airline_sentiment
0	@VirginAmerica What @dhepburn said.	neutral
1	@VirginAmerica plus you've added commercials t...	positive
2	@VirginAmerica I didn't today... Must mean I n...	neutral
3	@VirginAmerica it's really aggressive to blast...	negative
4	@VirginAmerica and it's a really big bad thing...	negative

In [26]:
```
# B) Tokenize - lemmatizer and Stemmer are fed tokens not strings, hence we can perform a few  more pre-processing tasks
                # as string inputs, before tokenizing later
```

In [27]:
```
# replace contractions with full words - this is important since we are checking sentiment, and need to later filter out the
# stop words we want to retain

def replace_contractions (text) :
    return contractions.fix(text)

data1['text']=data1['text'].apply(lambda x: replace_contractions(x))

data1.head()
```

Out[27]:

	text	airline_sentiment
0	@VirginAmerica What @dhepburn said.	neutral
1	@VirginAmerica plus you have added commercials...	positive
2	@VirginAmerica I did not today... Must mean I ...	neutral
3	@VirginAmerica it is really aggressive to blas...	negative
4	@VirginAmerica and it is a really big bad thin...	negative

In [ ]:

In [28]:
```
# as we can see in line 2 "didn't" has been changed to did not ... Contractions library worked
```

In [29]:
```
# C) remove the  numbers from text as it adds no value to sentiment analysis
def remove_numbers(text):
    text=re.sub(r'\d+','',text)
    return text

data1['text']=data1['text'].apply(lambda x:remove_numbers(x))
```

In [30]:
```
data1.head()
```

Out[30]:

	text	airline_sentiment
0	@VirginAmerica What @dhepburn said.	neutral
1	@VirginAmerica plus you have added commercials...	positive
2	@VirginAmerica I did not today... Must mean I ...	neutral
3	@VirginAmerica it is really aggressive to blas...	negative
4	@VirginAmerica and it is a really big bad thin...	negative

In [31]:
```
# convert accented characters and turn them to regular English letters
# unicode library already imported above

def remove_accented(text):
    English_text=unicodedata.normalize('NFKD', text).encode('ascii','ignore').decode('utf-8','ignore')
    # text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
    return English_text

data1['text']=data1['text'].apply(lambda x:remove_accented(x))
```

In [ ]:

In [32]:
```
# D) Remove Special Characters and Punctuations
                            # while we already removed numbers specifically as instructed, we can
                            # also remove them using this user defined function as an option)

def remove_special_char_punctuations ( text,remove_digits=False) :
    pattern=r'[^a-zA-Z0-9\s]' if not remove_digits else r'[^a-zA-Z\s]'
    text=re.sub(pattern,'',text)
    return text

   #the ^ makes it NOT alphanumberc, hence letters and numbers if remove_digits is true
    # else just letters  r'[^a-zA-Z\s]'  if remove_digits is false

data1['text']=data1['text'].apply(lambda x:remove_special_char_punctuations(x,remove_digits=True))
```

In [33]:
```
data1.head()
# clearly the @ symbol and other characters have been removed  as seen below by checking the data visually
```

Out[33]:

	text	airline_sentiment
0	VirginAmerica What dhepburn said	neutral
1	VirginAmerica plus you have added commercials ...	positive
2	VirginAmerica I did not today Must mean I need...	neutral
3	VirginAmerica it is really aggressive to blast...	negative
4	VirginAmerica and it is a really big bad thing...	negative

In [34]:
```
# E) change all words  to lowecase
    # lower and upper case have different values and hence would be treated as different words
   # there is absolutely no need for additional features for such words, solve by lowercasing all

def to_lowercase (text):
    text=text.lower()
    return text

data1['text']=data1['text'].apply(lambda x:to_lowercase(x))
```

In [35]:
```
data1.head()
# lowercase worked well as we can see below
```

Out[35]:

	text	airline_sentiment
0	virginamerica what dhepburn said	neutral
1	virginamerica plus you have added commercials ...	positive
2	virginamerica i did not today must mean i need...	neutral
3	virginamerica it is really aggressive to blast...	negative
4	virginamerica and it is a really big bad thing...	negative

In [ ]:

In [36]:
```
#B) Tokenize in preparation for the Lemmanizing

def text_tokenizer(text):
    tokenizer=ToktokTokenizer()
    text=tokenizer.tokenize(text)
    return text

data1['text']=data1['text'].apply(lambda x: text_tokenizer(x))
```

In [37]:
```
data1.head()
```

Out[37]:

	text	airline_sentiment
0	[virginamerica, what, dhepburn, said]	neutral
1	[virginamerica, plus, you, have, added, commer...	positive
2	[virginamerica, i, did, not, today, must, mean...	neutral
3	[virginamerica, it, is, really, aggressive, to...	negative
4	[virginamerica, and, it, is, a, really, big, b...	negative

In [ ]:

In [38]:
```
# DEFINE STOPWORD LIST
# BECAUSE WE ARE CHECKING FOR SENTIMENT WE MUST REMOVE KEY STOPWORDS FROM LIST THAT CHANGE MEANING
# SUCH AS NOT, DOES NOT, NO ...

stopwords=stopwords.words('english')
# stopwords already specifically imported in with libriaries above
```

In [39]:
```
print(stopwords)
```

```
['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]:
```
# insure meaning is not lost since we are checking for sentiment and words like no or not will
  # reverse meaning
keep_stopwords=['no','not','nor','ma'] # contractions has already removed/changed all the xx't words
                                      # hence they are not present in the list, and do not need to be
                                      # added to this custom list
```

In [41]:
```
# customize stopwords list
stopwords = list(set(stopwords) - set(keep_stopwords))
print (stopwords)
```

```
['very', 'both', 'his', 'too', 'themselves', 'wouldn', 'because', 'of', 'yours', 'these', 'can', 'they', 'll', "don't", 'some', 'just', 'ourselves', "it's", 'my', 'was', "needn't", "hadn't", "you'll", 'being', 'ain', 'above', 'yourself', 'between', 'yourselves', 'should', 'couldn', 'by', 't', 'from', 'own', 'she', 'into', 'why', 'now', 'in', 'than', "won't", 'other', 'where', 'were', 'until', 'a', 'having', 'then', 'all', 'there', 'which', 'be', 'd', 'he', 'herself', "doesn't", 'through', 'it', 'isn', 'i', 'while', "you're", 'did', "hasn't", "wouldn't", 'here', 'most', 'when', "shouldn't", 'theirs', 'under', 'is', 'after', "you've", 'your', 's', 're', 'ours', 'shan', 'an', "should've", 'such', 'you', 'off', 'hadn', "couldn't", 'has', 'hasn', 'once', 'up', 'aren', "haven't", 'for', "didn't", 'its', 'do', 'further', 'didn', "that'll", 'had', 'does', 'her', 'don', 'our', 'them', 'needn', "mustn't", 'during', 'more', "shan't", 'have', 'those', 'mustn', 've', 'that', 'the', 'any', 'few', 'myself', "aren't", 'each', 'same', "she's", 'about', 'only', "isn't", 'weren', 'been', 'him', 'their', 'with', 'before', "mightn't", 'haven', 'against', 'this', 'y', 'on', 'won', 'as', 'if', 'over', 'o', 'doesn', 'again', 'will', 'hers', 'm', "weren't", 'am', 'are', 'and', 'whom', 'but', 'wasn', 'me', 'mightn', 'who', 'shouldn', 'what', 'below', 'himself', 'or', 'we', 'how', 'down', 'at', "you'd", 'to', 'out', 'so', 'itself', "wasn't", 'doing']
```

In [ ]:

In [42]:
```
# remove stopwords from dataset

def remove_stopwords (words):
    new_words=[]
    for word in words:
        if word not in stopwords:
            new_words.append(word)
    return new_words

data1['text']=data1['text'].apply(lambda x: remove_stopwords(x))
```

In [45]:
```
# F)  Lemmatize
# Lemanizer is a better tool as it checks if the word exists in a dictionary, but down side is
# that its slower on than stemmer, hence the time difference would be applicable in analysing large
# data sets. Since we are using a relatively small dataset of less than 15,000 records,  lets choose
# Lemanizer over stemmer because its a more accurate tool expected to produce better results

#nltk.download('wordnet')                         # this is already on top and imported
# from nltk.stem.wordnet import WordNetLemmatizer  # I left it here for reference



lemmatizer=WordNetLemmatizer()

def lemmatize_words (text):
    new_words=[]
    for word in text:
        new_words.append(lemmatizer.lemmatize(word, pos='v'))
    return new_words

data1['text']=data1['text'].apply(lambda x:lemmatize_words(x))
```

In [46]:
```
data1.head()
```

Out[46]:

	text	airline_sentiment
0	[virginamerica, dhepburn, say]	neutral
1	[virginamerica, plus, add, commercials, experi...	positive
2	[virginamerica, not, today, must, mean, need, ...	neutral
3	[virginamerica, really, aggressive, blast, obn...	negative
4	[virginamerica, really, big, bad, thing]	negative

In [47]:
```
# G) join words back to strings in preparation of Vectorization

def join_words(text):
    return ' '.join(text)

data1['text']=data1['text'].apply(lambda x:join_words(x))
```

In [48]:
```
# H) print first 5 rows of data after pre-processing
data1.head(5)
```

Out[48]:

	text	airline_sentiment
0	virginamerica dhepburn say	neutral
1	virginamerica plus add commercials experience ...	positive
2	virginamerica not today must mean need take an...	neutral
3	virginamerica really aggressive blast obnoxiou...	negative
4	virginamerica really big bad thing	negative

Vectorization

In [49]:
```
# 4A) Using CountVectorizer to converta text to numbers

from sklearn.feature_extraction.text import CountVectorizer  # import the needed tools

vectorizer=CountVectorizer (max_features=1000) # limit ourselves to 1000 word vocabulary in interest
                                           # of computational time (dimentionality curse management)

vocabulary=vectorizer.fit(data['text']) # create a vocabulary list of 1000 most frequent words,
                                        # with mapped unique numbers

vector=vectorizer.transform(data['text']) # change the data to matrix form, and place values in matrix

vector=vector.toarray() # convert vector to an an array, a format we can feed our classifier later
```

In [84]:
```
print (vector.shape)
print (type(vector))
print()
print(vector)
```

```
(14640, 1000)
<class 'numpy.ndarray'>

[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 1 0 0]
 [0 0 0 ... 0 0 0]]
```

In [50]:
```
print (vectorizer.vocabulary_) # visual check of mapping numbers to 1000 words - our vocabulary
```

```
{'virginamerica': 930, 'what': 956, 'said': 746, 'plus': 677, 'you': 996, 've': 925, 'to': 873, 'the': 850, 'experience': 305, 'didn': 257, 'today': 87
4, 'must': 592, 'mean': 556, 'need': 597, 'take': 834, 'another': 64, 'trip': 889, 'it': 472, 'really': 707, 'entertainment': 292, 'in': 461, 'your': 9
97, 'amp': 61, 'they': 856, 'have': 413, 'little': 520, 'and': 63, 'big': 124, 'bad': 108, 'thing': 857, 'about': 28, 'seriously': 770, 'would': 984,
'pay': 656, '30': 17, 'flight': 334, 'for': 354, 'seats': 759, 'that': 849, 'this': 860, 'only': 632, 'flying': 348, 'yes': 993, 'every': 298, 'time':
870, 'fly': 346, 'won': 972, 'go': 384, 'away': 103, 'missed': 577, 'without': 971, 'there': 854, 'https': 448, 'co': 190, 'well': 953, 'but': 145, 'no
w': 615, 'do': 264, 'was': 941, 'amazing': 57, 'arrived': 85, 'an': 62, 'hour': 440, 'early': 282, 're': 702, 'too': 879, 'good': 390, 'me': 555, 'di
d': 256, 'know': 488, 'is': 468, 'second': 760, 'cause': 165, 'of': 619, '10': 1, '24': 14, 'lt': 538, 'pretty': 688, 'so': 794, 'much': 590, 'better':
121, 'than': 846, 'such': 821, 'great': 392, 'deal': 239, 'already': 53, 'my': 593, '2nd': 16, 'haven': 414, 'even': 295, 'gone': 388, 'on': 628, '1s
t': 8, 'yet': 995, 'again': 38, 'all': 49, 'from': 366, 'travel': 883, 'http': 447, 'thanks': 848, 'sfo': 775, 'schedule': 755, 'still': 815, 'mia': 56
4, 'excited': 303, 'first': 327, 'country': 221, 'lax': 500, 'mco': 554, 'heard': 418, 'nothing': 614, 'things': 858, 'virgin': 929, 'flew': 333, 'ny
c': 618, 'last': 498, 'week': 950, 'couldn': 218, 'sit': 787, 'seat': 757, 'due': 278, 'two': 900, 'either': 284, 'help': 421, 'be': 113, 'awesome': 10
4, 'bos': 135, 'fll': 343, 'please': 675, 'want': 938, 'with': 970, 'why': 963, 'are': 79, 'may': 552, 'over': 644, 'three': 864, 'times': 871, 'more':
585, 'other': 640, 'when': 957, 'available': 101, 'select': 765, 'love': 536, 'feel': 320, 'will': 966, 'making': 547, 'gt': 395, 'las': 497, 'non': 60
9, 'stop': 816, 'soon': 800, 'guys': 398, 'up': 910, 'seating': 758, 'friends': 365, 'gave': 375, 'free': 360, 'status': 813, 'been': 115, 'weeks': 95
2, 'called': 151, 'no': 608, 'response': 728, 'happened': 406, 'ur': 916, 'food': 353, 'options': 635, 'at': 93, 'least': 503, 'say': 752, 'site': 788,
'able': 27, 'anything': 71, 'next': 605, 'hrs': 446, 'fail': 311, 'miss': 576, 'don': 269, 'we': 946, 'll': 522, 'together': 875, 'very': 927, 'can': 1
55, 'get': 376, 'any': 68, 'cold': 192, 'air': 42, 'ewr': 301, 'middle': 566, 'hi': 428, 'just': 480, 'cool': 214, 'birthday': 125, 'add': 33, 'name':
595, 'during': 279, 'booking': 134, 'problems': 693, 'hours': 441, 'club': 189, 'online': 631, 'left': 506, 'iad': 453, 'one': 630, 'answering': 66, 'n
umber': 616, 'return': 729, 'phone': 663, 'call': 150, 'use': 921, 'service': 771, 'option': 634, 'news': 604, 'could': 217, 'start': 811, 'flights': 3
42, 'by': 148, 'end': 290, 'year': 991, 'via': 928, 'nice': 606, 'rt': 738, 'takeoff': 836, 'way': 945, 'best': 120, 'ever': 297, 'done': 270, 'airlin
e': 44, 'around': 82, 'down': 273, 'book': 132, 'support': 827, 'not': 612, 'working': 977, 'sign': 785, 'beyond': 123, 'hey': 427, 'flyer': 347, 'havi
ng': 415, 'hard': 409, 'getting': 378, 'account': 31, 'plz': 678, 'upgrade': 914, 'ticket': 868, 'moved': 589, 'new': 602, 'city': 183, 'where': 958,
'how': 443, 'before': 116, 'leaving': 505, 'dallas': 232, 'feb': 317, 'reason': 708, 'rock': 733, 'wow': 986, 'mind': 571, 'after': 36, 'night': 607,
'think': 859, 'were': 955, 'supposed': 828, 'off': 620, 'minutes': 575, 'ago': 41, 'website': 949, 'shows': 784, 'though': 862, 'wish': 969, 'out': 64
3, 'atlanta': 95, 'la': 491, 'lga': 512, 'trying': 893, 'since': 786, 'page': 648, 'never': 600, 'thx': 867, 'lady': 493, 'she': 777, 'let': 509, 'us':
917, 'sorry': 801, 'had': 399, 'dca': 238, 'tried': 888, 'check': 176, 'through': 865, 'someone': 797, 'hold': 432, '40': 20, '50': 22, 'earlier': 281,
'tonight': 878, '11': 3, 'award': 102, 'everything': 300, 'fine': 326, 'until': 909, 'lost': 532, 'bag': 109, 'change': 169, 'reservation': 723, 'credi
t': 225, 'card': 160, 'fee': 318, 'or': 636, 'customer': 229, 'team': 841, 'if': 458, 'booked': 133, 'baggage': 110, 'needs': 599, 'its': 474, 'ride':
730, 'plane': 671, 'crew': 226, 'airlines': 45, 'should': 780, 'like': 514, 'yall': 989, 'sat': 749, 'morning': 586, 'correct': 215, 'watch': 943, 'som
e': 796, '35': 18, '000': 0, 'different': 258, 'policy': 682, 'media': 559, 'bags': 111, 'going': 386, 'anyway': 72, 'speak': 805, 'human': 450, 'asa
p': 88, 'thank': 847, 'traveler': 884, 'southwestair': 803, 'jetblue': 476, 'then': 853, 'once': 629, 'member': 560, 'im': 459, '100': 2, 'delayed': 24
2, 'late': 499, 'cancelled': 156, 'four': 359, 'which': 959, 'business': 143, 'trouble': 890, 'wife': 964, 'bring': 139, 'using': 923, 'code': 191, 'ha
s': 410, 'share': 776, 'life': 513, 'happens': 407, 'am': 56, 'home': 434, 'back': 107, 'yeah': 990, 'points': 681, 'most': 587, 'tv': 896, 'disappoint
ed': 260, 'flightled': 340, 'went': 954, 'jfk': 477, 'saturday': 750, 'landed': 495, 'here': 426, 'friendly': 364, 'btw': 141, 'isn': 469, 'both': 137,
'mobile': 579, 'passengers': 654, 'leave': 504, 'told': 876, 'their': 851, 'class': 185, 'find': 325, 'who': 961, 'anyone': 70, 'doing': 267, 'direct':
259, 'layover': 501, 'vegas': 926, 'bought': 138, 'people': 658, 'same': 747, 'customerservice': 231, 'line': 515, 'hung': 451, 'info': 463, 'schedule
d': 756, 'changed': 170, 'weather': 947, 'looks': 529, 'lots': 534, 'come': 193, 'phl': 661, 'horrible': 438, 'flown': 344, 'easy': 283, 'helpful': 42
3, 'rep': 716, 'front': 367, 'right': 732, 'running': 742, 'gate': 374, 'waited': 935, 'kept': 484, '2015': 11, 'doesn': 266, 'totally': 882, 'folks':
349, 'problem': 692, 'min': 570, 'delay': 241, 'connecting': 209, 'seems': 763, 'long': 525, 'san': 748, 'provide': 695, 'wait': 934, 'calling': 152,
'completely': 202, 'month': 583, 'depart': 248, 'customers': 230, 'because': 114, 'process': 694, 'does': 265, 'link': 517, 'tsa': 894, 'pre': 687, 'te
rrible': 845, 'hotel': 439, 'assistance': 92, 'yesterday': 994, 'our': 642, 'give': 379, 'longer': 526, 'pls': 676, 'dropped': 277, 'always': 55, 'bu
y': 146, 'them': 852, 'frustrated': 368, 'paying': 657, 'extra': 308, 'luggage': 540, 'might': 567, 'world': 979, 'takes': 837, 'flt': 345, 'monday': 5
81, 'cant': 158, 'as': 87, 'web': 948, 'staff': 809, 'super': 825, 'paid': 649, 'offer': 621, 'sad': 744, 'question': 698, 'possible': 684, 'under': 90
4, 'giving': 382, 'him': 430, 'dc': 237, 'work': 975, 'understand': 905, 'dm': 263, 'answer': 65, 'kids': 485, 'priority': 690, 'boarding': 131, 'check
ing': 179, 'tickets': 869, 'happy': 408, 'coming': 194, '23': 13, 'dfw': 255, 'friday': 362, 'keeps': 483, 'error': 294, 'contact': 212, 'minute': 574,
'reschedule': 721, 'fix': 328, 'got': 391, 'checked': 177, 'email': 286, 'tomorrow': 877, 'unacceptable': 903, 'into': 467, 'flighted': 336, 'stuck': 8
19, 'offered': 622, 'look': 527, 'agent': 39, 'closed': 186, 'plans': 673, 'austin': 99, 'route': 736, 'reply': 717, 'airport': 46, 'waiting': 936, 'ch
eckin': 178, 'desk': 250, 'open': 633, 'luv': 541, 'show': 782, 'united': 907, 'follow': 350, 'many': 549, 'worse': 981, 'respond': 726, 'spend': 806,
'money': 582, 'stranded': 818, 'landing': 496, 'choice': 182, 'southwest': 802, 'days': 236, 'confirmation': 206, 'claim': 184, 'rebook': 709, 'gold':
387, 'lot': 533, 'being': 118, 'see': 761, 'worst': 982, 'wrong': 987, 'issue': 470, 'missing': 578, 'americanairlines': 60, 'lose': 530, 'suck': 822,
'flightlation': 338, 'boston': 136, 'guy': 397, 'high': 429, 'quick': 699, 'apparently': 76, 'sitting': 789, 'sent': 769, 'keep': 481, 'ny': 617, 'pilo
ts': 668, 'job': 478, 'snow': 793, 'area': 80, 'afternoon': 37, 'row': 737, 'mechanical': 558, 'handle': 402, 'drink': 274, 'twitter': 899, 'charged':
```

```
174, 'refund': 714, 'access': 30, 'passbook': 652, 'received': 711, 'broken': 140, 'looking': 528, 'forward': 357, 'rescheduled': 722, 'gonna': 389, 'l
and': 494, 'cost': 216, '800': 24, 'report': 718, 'bit': 126, 'attendant': 96, 'something': 798, 'departure': 249, 'fun': 371, 'updates': 913, 'helpin
g': 424, 'while': 960, 'passenger': 653, 'shouldn': 781, 'he': 416, 'destination': 252, 'wasn': 942, 'itinerary': 473, 'march': 550, 'entire': 293, 'wi
ndow': 967, 'confirmed': 207, 'treat': 886, 'wtf': 988, 'frequent': 361, 'inconvenience': 462, 'full': 370, 'place': 669, 'instead': 465, 'makes': 546,
'cabin': 149, 'behind': 117, 'pilot': 667, 'tell': 842, 'past': 655, 'husband': 452, 'came': 154, 'despite': 251, 'gives': 381, 'delays': 243, 'few': 3
22, 'says': 754, 'expect': 304, 'safety': 745, 'day': 235, 'wanted': 939, 'newark': 603, 'hope': 435, 'probably': 691, 'board': 129, 'standby': 810, 'h
elped': 422, 'absolutely': 29, 'ready': 705, 'old': 627, 'less': 508, 'half': 401, 'price': 689, 'round': 735, 'fare': 315, 'made': 542, 'evening': 29
6, 'taking': 838, 'care': 161, 'her': 425, 'mileage': 568, 'several': 774, 'americanair': 59, 'glad': 383, 'took': 880, 'years': 992, 'try': 892, 'purc
hase': 696, 'hoping': 437, 'forced': 355, 'small': 792, 'carry': 162, 'empty': 289, 'space': 804, 'spoke': 808, 'loyal': 537, '3rd': 19, 'course': 222,
'family': 313, 'ok': 625, 'currently': 227, 'car': 159, 'sucks': 823, 'cust': 228, 'child': 181, 'chance': 168, 'joke': 479, 'also': 54, 'tix': 872, 'y
r': 998, 'send': 766, 'submitted': 820, 'request': 720, 'ceo': 167, 'planes': 672, 'thought': 863, 'computer': 203, '15': 5, 'cannot': 157, 'those': 86
1, 'guess': 396, 'make': 545, 'address': 34, 'app': 75, 'set': 773, 'system': 833, 'charge': 173, 'learn': 502, 'tweet': 897, 'between': 122, 'housto
n': 442, 'finally': 324, 'future': 373, 'poor': 683, 'each': 280, 'excellent': 302, '1k': 7, 'group': 394, 'room': 734, 'sure': 829, '12': 4, 'shit': 7
78, 'stay': 814, 'man': 548, 'wifi': 965, 'saying': 753, 'saw': 751, 'point': 680, 'part': 650, 'whole': 962, 'short': 779, 'started': 812, 'overhead':
645, 'person': 659, 'calls': 153, 'ask': 89, 'flighting': 337, 'luck': 539, 'lol': 524, 'others': 641, 'storm': 817, 'complete': 201, 'fees': 321, 'phi
lly': 660, 'months': 584, 'update': 911, 'reps': 719, 'actually': 32, 'word': 974, 'lack': 492, 'ord': 637, 'connections': 211, 'international': 466,
'orlando': 639, '90': 25, 'mins': 573, 'boarded': 130, 'issues': 471, 'domestic': 268, 'oh': 624, 'ridiculous': 731, 'case': 163, 'worked': 976, 'wonde
rful': 973, 'voucher': 932, 'mail': 543, 'company': 197, 'maybe': 553, 'pass': 651, 'son': 799, 'given': 380, 'miami': 565, 'drive': 275, 'own': 647,
'explain': 306, 'talk': 839, 'almost': 52, 'ppl': 686, 'daughter': 234, 'flightr': 341, 'anymore': 69, 'platinum': 674, 'run': 741, 'agents': 40, 'rud
e': 739, 'overnight': 646, 'hr': 445, 'iah': 454, 'taken': 835, 'used': 922, 'details': 254, 'charlotte': 175, 'form': 356, 'vouchers': 933, 'anywher
e': 73, 'ground': 393, '22': 12, 'else': 285, 'appreciate': 77, 'tarmac': 840, 'enough': 291, 'denver': 247, 'put': 697, 'found': 358, 'supervisor': 82
6, 'connect': 208, 'arriving': 86, 'thru': 866, 'connection': 210, 'upset': 915, 'services': 772, 'employees': 288, 'mom': 580, 'miles': 569, 'america
n': 58, 'rather': 700, 'telling': 843, 'delta': 245, 'attendants': 97, 'seem': 762, 'believe': 119, 'bc': 112, 'record': 712, 'these': 855, 'asking': 9
1, 'read': 704, 'everyone': 299, 'ua': 901, 'aircraft': 43, 'knew': 487, 'winter': 968, 'reach': 703, 'hear': 417, 'baby': 106, 'mine': 572, 'apology':
74, 'goes': 385, 'zero': 999, 'date': 233, 'members': 561, 'weekend': 951, 'asked': 90, 'flightd': 335, 'held': 419, 'caused': 166, '25': 15, 'drop': 2
76, 'message': 563, 'complaint': 199, 'awful': 105, 'jet': 475, 'matter': 551, 'atl': 94, 'traveling': 885, 'works': 978, 'worth': 983, 'airports': 47,
'conf': 204, 'communication': 195, 'far': 314, 'note': 613, 'top': 881, 'unitedairlines': 908, 'kudos': 490, 'advisory': 35, 'extremely': 309, 'real':
706, 'resolved': 725, 'clothes': 187, 'leg': 507, 'chicago': 180, 'lounge': 535, 'okay': 626, 'load': 523, 'terminal': 844, 'arrival': 83, '20': 9, 'in
formation': 464, 'confirm': 205, 'vacation': 924, 'followed': 351, 'nope': 611, 'sleep': 791, 'updated': 912, 'ruined': 740, 'compensation': 198, 'hi
s': 431, 'original': 638, 'happen': 405, 'delivered': 244, 'arrive': 84, 'employee': 287, 'knows': 489, 'answers': 67, 'feedback': 319, 'offering': 62
3, 'keeping': 482, 'clt': 188, 'move': 588, 'idea': 457, 'neveragain': 601, 'responding': 727, 'frustrating': 369, 'figure': 323, 'counter': 219, 'unfo
rtunately': 906, 'phoenix': 662, 'fact': 310, 'wouldn': 985, 'needed': 598, 'means': 557, 'bwi': 147, 'pm': 679, 'fixed': 329, 'unable': 902, 'ice': 45
5, 'post': 685, 'phones': 664, 'letting': 511, 'den': 246, 'fleet': 332, 'spent': 807, 'situation': 790, 'literally': 519, 'list': 518, 'changes': 171,
'200': 10, 'however': 444, 'control': 213, 'aa': 26, 'catch': 164, 'reservations': 724, 'reflight': 713, 'dealing': 240, '45': 21, 'seen': 764, 'tuesda
y': 895, 'id': 456, 'multiple': 591, 'maintenance': 544, 'friend': 363, 'twice': 898, 'allowed': 51, '1hr': 6, 'counting': 220, 'myself': 594, 'rebooke
d': 710, 'sunday': 824, 'sense': 768, 'aren': 81, 'hate': 412, 'huge': 449, 'hopefully': 436, 'complaints': 200, 'rdu': 701, 'usairways': 919, 'usairwa
ysfail': 920, 'door': 272, 'pick': 666, '75': 23, 'crazy': 224, 'airways': 48, 'showing': 783, 'gets': 377, 'failed': 312, 'treated': 887, 'fault': 31
6, 'sending': 767, 'losing': 531, 'lines': 516, 'nashville': 596, 'changing': 172, 'attitude': 98, 'switch': 832, 'cover': 223, 'none': 610, 'flightlat
ions': 339, 'explanation': 307, 'dont': 271, 'appreciated': 78, 'hang': 403, 'social': 795, 'plan': 670, 'following': 352, 'runway': 743, 'hasn': 411,
'bumped': 142, 'true': 891, 'live': 521, 'busy': 144, 'allow': 50, 'funeral': 372, 'hello': 420, 'relations': 715, 'mess': 562, 'hangs': 404, 'disconne
cted': 261, 'water': 944, 'bna': 128, 'kind': 486, 'phx': 665, 'volume': 931, 'automated': 100, 'companion': 196, 'haha': 400, 'sw': 830, 'fl': 330, 'l
etter': 510, 'worries': 980, 'warm': 940, 'holding': 433, 'swa': 831, 'destinationdragons': 253, 'imaginedragons': 460, 'usair': 918, 'blue': 127, 'fle
ek': 331, 'wall': 937, 'dividend': 262}
```


In [51]:
```
# Label
labels=data['airline_sentiment']
type(labels)
```

Out[51]:
```
pandas.core.series.Series
```

In [ ]:
```
# 4B) Do the same operation using TDI_IDF Vectorizer
```

In [65]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf= TfidfVectorizer(max_features=1000) # as before  limit vocabulary to 1000 most
                                                     # frequent words

vocabulary_tfidf=vectorizer_tfidf.fit(data['text']) # create a vocabulary list of 1000 most frequent words,
                                                # with mapped uniqure numbers
print('IDF values are :')
print (vectorizer_tfidf.idf_)

vector_tfidf=vectorizer.transform(data['text']) #change the data to matrix form
                                                  # and place values in matrix

vector_tfidf=vector_tfidf.toarray()  # convert vector to an an array
```

```
IDF values are :
[7.15759389 5.39862424 6.60259704 6.26084775 6.53113808 5.97646057
 7.29574423 7.06522057 6.01687011 5.82089647 6.51404365 6.98066318
 7.22428526 7.19038371 6.17274048 6.37207339 6.69976079 5.40979754
 7.25937658 7.22428526 6.24777567 6.12567297 6.24777567 7.29574423
 6.43269801 7.19038371 5.26370492 5.79579055 4.38904557 7.15759389
 6.92801945 6.30112165 6.12567297 6.17274048 6.76293969 7.37270527
 4.44954369 7.15759389 4.64616048 5.02323659 5.56770057 5.68630631
 5.63575403 7.09507353 4.60764481 5.54172508 4.74803667 7.41352726
 6.13723379 4.11460873 7.00806215 7.06522057 6.0272329  5.39862424
 5.70123196 5.90944986 4.52779588 5.99646124 6.23487226 2.60100421
 7.19038371 4.16833413 3.74039616 2.48115385 5.00433243 5.91875226
 7.06522057 7.15759389 4.52084336 7.25937658 5.85538264 5.67892621
 7.22428526 7.25937658 6.69976079 5.94719019 6.92801945 5.8379909
 6.85391147 3.64845867 7.15759389 6.69976079 6.40192635 7.12584519
 6.62128918 6.54852982 7.29574423 4.57298788 6.83038098 6.69976079
 6.41719382 6.74143349 6.58424791 3.36537208 6.90270164 6.98066318
 6.08072158 6.54852982 7.19038371 6.90270164 6.87800902 5.928142
 6.83038098 6.26084775 5.77939674 6.51404365 7.33348455 4.35130525
 5.40419529 4.49575653 5.17993504 4.98946227 6.37207339 3.53799536
 4.84218811 3.85106173 5.24924684 7.29574423 5.28827618 6.54852982
 5.33408572 5.39308406 6.76293969 6.87800902 6.30112165 7.12584519
 7.25937658 6.74143349 7.15759389 5.90023321 6.74143349 5.40979754
 5.40979754 5.40419529 5.60797447 6.0589816  6.11424428 6.30112165
 6.7849186  6.74143349 6.40192635 7.29574423 7.33348455 5.82940716
 6.67955809 3.48661564 6.7849186  6.60259704 4.63574372 7.00806215
 4.40949618 5.66432741 6.20955446 6.58424791 7.06522057 3.25464418
 3.66402318 6.64033737 6.83038098 6.37207339 6.22213324 5.59436882
 6.65975546 6.72038008 7.29574423 6.95399493 7.37270527 6.65975546
 6.24777567 4.88447083 6.38688847 6.64033737 7.03623303 6.32890121
 7.00806215 6.43269801 4.90800132 5.82940716 6.95399493 6.69976079
 6.17274048 7.29574423 7.29574423 6.76293969 6.0589816  5.78756005
 7.22428526 6.98066318 6.08072158 6.7849186  3.52085698 7.22428526
 7.00806215 5.84664896 6.31491497 6.74143349 6.87800902 6.12567297
 6.95399493 6.72038008 7.33348455 7.12584519 6.95399493 6.7849186
 7.25937658 7.15759389 6.35747459 6.85391147 7.22428526 6.08072158
 5.56114317 6.98066318 6.04828631 7.29574423 6.41719382 6.87800902
 6.48070723 5.06612815 6.01687011 6.83038098 6.64033737 7.09507353
 7.22428526 7.37270527 7.06522057 5.85538264 5.16223546 6.7849186
 6.64033737 3.98088505 5.37122527 6.95399493 6.09177142 6.92801945
 7.25937658 4.81702955 5.17993504 6.64033737 5.87308222 6.74143349
 7.19038371 4.9251544  4.32627988 5.5809458  7.00806215 5.928142
 6.58424791 6.18486184 7.25937658 6.08072158 6.34308585 7.06522057
 6.37207339 6.17274048 6.90270164 5.52267689 4.81392877 5.28331339
 6.10294472 6.12567297 5.96660828 6.95399493 7.19038371 4.9821093
 3.82079167 5.38209494 5.4795933  6.22213324 7.33348455 4.46253088
 5.73176869 7.15759389 7.25937658 5.67160017 7.37270527 6.80739146
 7.19038371 7.37270527 5.10264337 6.49723653 6.72038008 6.06979251
 6.0377042  6.92801945 6.83038098 6.48070723 5.15350178 6.95399493
 6.13723379 6.76293969 6.62128918 6.30112165 7.33348455 6.76293969
 6.62128918 4.91140848 7.22428526 5.12352095 5.86419327 6.11424428
 6.58424791 6.04828631 7.22428526 7.25937658 6.64033737 5.29827627
```

```
7.03623303 7.33348455 6.00661361 7.06522057 7.00806215 5.99646124
7.29574423 6.12567297 6.24777567 7.03623303 6.87800902 7.00806215
6.35747459 7.25937658 6.5662294  6.74143349 6.43269801 6.90270164
5.76326735 5.53533529 6.92801945 5.02706068 6.0272329  7.09507353
7.22428526 5.5809458  5.56114317 6.76293969 2.46447191 7.33348455
5.19795354 6.80739146 7.03623303 6.85391147 4.37697299 5.51017673
4.14904092 6.49723653 6.95399493 5.66432741 4.66732529 7.25937658
4.8453779  7.19038371 5.71638377 7.29574423 6.95399493 6.26084775
2.40233657 7.29574423 6.76293969 6.67955809 6.20955446 7.09507353
5.72404664 7.37270527 7.12584519 6.46444671 7.06522057 6.98066318
3.54145857 7.06522057 6.31491497 6.53113808 6.09177142 7.03623303
7.37270527 6.85391147 4.4074322  6.287516   3.43696573 6.98066318
5.11511754 5.46168238 6.48070723 7.03623303 7.00806215 6.64033737
4.88447083 7.12584519 4.81392877 7.12584519 7.33348455 7.00806215
4.89784895 4.48233351 4.82638999 6.13723379 6.7849186  6.11424428
6.44844636 6.74143349 4.67000267 4.3992186  7.15759389 6.287516
6.95399493 7.15759389 7.33348455 6.83038098 6.54852982 6.80739146
6.09177142 6.53113808 4.45169654 7.15759389 6.92801945 3.23913999
6.0272329  5.84664896 5.54815597 6.53113808 6.5662294  7.09507353
7.06522057 3.85224446 7.00806215 5.9568521  6.80739146 5.49171466
4.94260685 6.32890121 6.0272329  6.87800902 6.54852982 6.22213324
4.15703457 6.85391147 4.92862061 5.65710716 6.80739146 6.60259704
6.16076429 5.62176779 4.42616324 4.12855163 6.44844636 4.12855163
7.29574423 6.35747459 5.21168374 3.56515428 6.58424791 6.98066318
6.74143349 6.09177142 7.09507353 6.46444671 6.60259704 7.09507353
7.12584519 6.51404365 4.17976282 6.54852982 6.74143349 2.83324762
7.12584519 5.6011485  6.58424791 5.98641091 6.5662294  5.50398476
2.74069843 6.00661361 5.56114317 5.78756005 2.93146677 7.41352726
5.72404664 6.64033737 2.87601156 5.30331406 6.13723379 6.65975546
3.72986975 5.67892621 7.37270527 7.19038371 7.29574423 6.40192635
7.00806215 7.12584519 4.7194633  7.37270527 7.15759389 6.80739146
6.53113808 7.33348455 7.12584519 6.19713194 6.48070723 6.65975546
4.88447083 4.58031392 5.62176779 7.15759389 7.19038371 6.13723379
5.94719019 6.22213324 5.43252579 6.85391147 6.40192635 5.15350178
7.25937658 7.25937658 6.49723653 6.53113808 4.54894826 5.40979754
7.22428526 6.38688847 6.83038098 7.19038371 6.37207339 6.83038098
5.02706068 7.19038371 6.53113808 5.40979754 6.46444671 5.96660828
5.86419327 6.41719382 6.76293969 7.00806215 5.11511754 6.48070723
7.09507353 7.12584519 5.10264337 6.64033737 7.29574423 6.98066318
5.08624956 6.98066318 5.25404301 7.37270527 6.60259704 4.86473334
6.43269801 5.94719019 7.25937658 5.93762074 7.22428526 7.25937658
6.34308585 6.23487226 6.67955809 3.12306782 6.54852982 7.00806215
6.34308585 6.95399493 6.58424791 7.00806215 7.25937658 6.32890121
7.00806215 6.60259704 7.06522057 6.27409298 7.12584519 5.54815597
5.39308406 7.33348455 7.15759389 5.79579055 6.58424791 4.96395998
5.52267689 5.70123196 5.85538264 7.19038371 7.15759389 6.60259704
6.04828631 6.80739146 6.80739146 4.6593359  5.49783089 6.20955446
7.29574423 6.90270164 5.17548069 7.06522057 6.98066318 2.64213667
7.03623303 6.12567297 6.87800902 4.30931434 6.67955809 6.23487226
4.83900845 7.25937658 5.01942706 6.40192635 7.15759389 5.14484372
5.67892621 5.5809458  3.36828541 6.69976079 7.33348455 7.15759389
3.28639288 7.25937658 5.53533529 3.6689372  5.14484372 7.22428526
```

```
6.24777567 3.01037126 4.91140848 6.38688847 6.92801945 7.22428526
6.13723379 5.928142   6.90270164 6.16076429 2.45564832 6.34308585
4.28713229 5.32372293 4.77149816 6.0377042  6.32890121 6.60259704
4.4074322  5.84664896 6.98066318 6.92801945 5.26370492 7.25937658
4.12233077 4.0362242  4.61777148 7.19038371 7.15759389 6.58424791
7.22428526 6.0377042  6.95399493 6.08072158 7.29574423 6.76293969
5.35513913 6.32890121 5.70123196 6.74143349 4.94967402 5.928142
6.62128918 5.8124576  7.29574423 4.55609966 7.15759389 6.43269801
6.62128918 6.06979251 6.72038008 6.7849186  7.15759389 4.18800689
6.01687011 6.90270164 7.00806215 4.27984628 6.48070723 6.51404365
7.29574423 6.92801945 6.27409298 6.60259704 6.26084775 6.12567297
6.40192635 7.33348455 7.15759389 7.06522057 6.83038098 6.85391147
7.06522057 6.95399493 5.90023321 5.30331406 6.65975546 6.76293969
6.87800902 5.62873646 6.38688847 6.62128918 6.80739146 7.06522057
4.68349815 6.72038008 6.98066318 6.92801945 6.49723653 4.8944876
6.30112165 5.80408935 5.65710716 6.11424428 6.92801945 6.60259704
5.62176779 7.06522057 6.0589816  6.27409298 7.15759389 7.09507353
6.62128918 6.95399493 7.12584519 5.49783089 6.34308585 7.33348455
6.80739146 7.15759389 5.26370492 6.32890121 7.25937658 6.08072158
5.23972296 7.09507353 6.72038008 7.09507353 7.09507353 6.5662294
6.0272329  5.70123196 7.33348455 6.7849186  7.03623303 6.46444671
6.95399493 6.92801945 5.41543136 5.54815597 6.22213324 6.54852982
7.29574423 7.22428526 5.87308222 6.43269801 5.62873646 7.03623303
6.51404365 5.09851965 7.33348455 5.31858153 6.64033737 5.01942706
6.83038098 6.51404365 7.00806215 7.33348455 6.0377042  7.00806215
7.09507353 5.37122527 6.19713194 3.74251481 7.09507353 6.95399493
7.06522057 5.78756005 7.03623303 5.2588623  7.09507353 7.22428526
4.79552334 7.00806215 6.14892983 7.19038371 7.33348455 7.25937658
5.4382895  6.31491497 6.10294472 5.31858153 6.90270164 6.83038098
7.19038371 6.22213324 3.83232582 7.22428526 5.3288909  5.10678416
6.09177142 7.29574423 6.38688847 5.97646057 6.0272329  2.78610647
7.09507353 6.00661361 7.19038371 6.74143349 7.12584519 5.36047247
6.64033737 6.20955446 7.33348455 5.88205089 6.44844636 4.24419188
6.04828631 7.06522057 6.13723379 5.44408661 7.25937658 6.54852982
6.72038008 6.54852982 6.7849186  6.87800902 6.87800902 6.87800902
6.09177142 5.40979754 6.74143349 6.65975546 6.98066318 5.62873646
5.01563199 6.7849186  7.37270527 7.09507353 6.22213324 6.09177142
6.14892983 5.91875226 5.64993867 6.51404365 6.49723653 5.96660828
5.09032288 4.21485414 3.61616716 3.20635017 2.11666929 5.26857111
5.10264337 4.83267932 4.16671207 6.37207339 4.05055109 6.32890121
6.46444671 5.54172508 3.39789527 6.18486184 6.04828631 6.69976079
6.35747459 5.21630268 6.5662294  5.93762074 5.08624956 5.73955083
3.96751586 5.44991753 7.15759389 1.83947389 4.55371017 6.35747459
5.01185127 4.86799599 5.51017673 5.14054264 6.08072158 6.92801945
7.15759389 5.12352095 7.41352726 6.46444671 6.95399493 7.15759389
5.62176779 5.2588623  7.29574423 7.12584519 6.09177142 4.75677035
6.80739146 6.95399493 6.74143349 6.24777567 6.37207339 6.18486184
5.22560508 6.53113808 7.09507353 6.09177142 6.80739146 5.99646124
7.09507353 2.32721776 6.74143349 5.84664896 4.00128004 5.91875226
7.37270527 6.80739146 6.01687011 6.92801945 6.30112165 4.13481144
6.69976079 2.58821803 7.29574423 5.59436882 6.48070723 6.37207339
6.26084775 4.44739546 6.13723379 5.03862151 5.928142   7.19038371
```

```
 4.34353822 7.37270527 5.91875226 7.22428526 4.82952971 6.51404365
 4.72228418 7.12584519 5.03475303 6.65975546 7.33348455 3.32825147
 6.09177142 7.25937658 7.19038371 4.74514232 3.56337966 4.87455339
 7.29574423 5.54815597 5.79579055 7.09507353 6.26084775 5.48563562
 6.32890121 5.03089946 3.90073881 4.12855163 5.37664533 5.91875226
 6.08072158 5.19795354 6.80739146 4.21315491 6.12567297 5.85538264
 3.97819687 7.41352726 7.06522057 6.72038008 3.30950743 5.91875226
 5.40419529 7.19038371 7.06522057 5.12352095 7.29574423 5.82940716
 7.12584519 6.69976079 7.25937658 6.48070723 5.12774929 7.19038371
 4.305583   6.60259704 6.76293969 6.26084775 6.87800902 7.12584519
 6.76293969 6.0272329  6.74143349 5.12352095 5.90023321 5.56114317
 2.37630413 3.21319738 7.29574423 6.98066318]
```

In [ ]:

In [66]:
```
print (vector_tfidf.shape)
print (type(vector_tfidf))
print()
print(vector_tfidf[0:1])
```

```
(14640, 1000)
<class 'numpy.ndarray'>

[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

CLASSIFICATION - FIT AND EVALUATE USING BOTH TYPES OF VECTORIZATION

In [ ]:

In [55]:
```
# split data into train and test set in preparation of builing and testing a model

from sklearn.model_selection import train_test_split

X_train,X_test, y_train, y_test = train_test_split(vector,labels,test_size=0.3,random_state=42)
```

In [ ]:

In [56]:
```
# classify CountVectorizer Bags of Words

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

forest=RandomForestClassifier(n_estimators=10,n_jobs=4)

forest=forest.fit(X_train,y_train)

print(forest)

print(np.mean(cross_val_score(forest,vector,labels,cv=10)))
```

```
RandomForestClassifier(n_estimators=10, n_jobs=4)
0.7118169398907103
```

In [57]:
```
# now apply the trained model on test data

result=forest.predict(X_test)
print (result)
```

```
['positive' 'negative' 'negative' ... 'negative' 'negative' 'negative']
```

In [71]:
```
Rf_score=forest.score(X_test,y_test)
print ('The accuracy for the test data is',Rf_score)
```

```
The accuracy for the test data is 0.7527322404371585
```

In [ ]:
```
# The accuracy of the model on test data from COUNTVECTORIZER is good - 75%
```

FIT AND EVALUATE TFIDF VECTORIZER DATA

In [67]:
```
# SPLIT THE  TDFIDF VECTORIZER DATA INTO TRAIN AND TEST
# SPLIT LIBRARY IMPORTED ABOVE  ALREADY



XX_train,XX_test,yy_train,yy_test=train_test_split(vector_tfidf,labels,test_size=0.3,random_state=42)

# used new test and train vars in case i am running again sub-sections of this worksheet
# and would like to keep values of vars unique rather than relod old vars with new data
```

In [75]:
```
# classify TDF-IDF Vectorizer

# libraries already imported above

forest_tfidf=RandomForestClassifier(n_estimators=10,n_jobs=4)

forest_tfidf=forest_tfidf.fit(XX_train,yy_train)

print(forest_tfidf)

print(np.mean(cross_val_score(forest_tfidf,vector_tfidf,labels,cv=10)))
```

```
RandomForestClassifier(n_jobs=4)
0.7289617486338797
```

In [77]:
```
# now apply the trained model on test data

result=forest.predict(XX_test)
print (result)
```

```
['positive' 'negative' 'negative' ... 'negative' 'negative' 'negative']
```

In [78]:
```
Rf_score_tfidf=forest.score(XX_test,yy_test)
print ('The accuracy for the TFIDF test data is',Rf_score_tfidf)
```

```
The accuracy for the TFIDF test data is 0.7527322404371585
```

In [ ]:
```
# Both models have scored high above 75% in test data and do not have significantly different results
```

In [ ]:
```
## COMMENT ON RESULTS
# If we need still a higher result, it can be improved by increasing the depth of the forest !!

# Another hyper variable we can use is to increase the number of words in our vocabulary from 1000 to more ...
```

6A) Summarize

Analysing text, or properly called Natural Language Processing NLP, is an important part of AI, since its applications are wide. Classifying the type of text, whether a document or inquiry, will catagorize it and place progress it to the right department for processing shortening time and human input, as well as classyfying documents into better organized and more usful data banks and resources, and lastly analysis of sentiment.

This last one is very important since it carries the inherent benefit of predictive power of AI.

Given the amout of data out there, tweeter opinions, blogs, newspapers, emails, chats, web site form inputs all this can be utilizied to calculate the mood (sentiment) towards really any subject.

Assuming that "majority rules" we can ask questions about anything such as : 1) Who will win the 2020 presidential election in USA ? 2) Will we have a stock market crash ? 3) Will there be Corona Virus vaccine ? 4) What will be the weather next week ?

The subject range does not have limits, yet we can process text, and classify opinions by calculating and assigning a clasification, weather binary such as negative or positive, agree or disagree, high medium low or hot and cold, wet or dry ..

In the end we can base our prediction on the opinion of the public, and arrive at an opinion based on our calculations of majority rules, (or set own threshold) and provide a prediction to the questions above based on sentiment.

In order to evaluate the text, we must first pre-process it. Hence its important to :

- get rid of any HTML tags left over from web scraping for data- they add absolutely no predictive value

- remove numbers, perhaps in most cases unless numbers themselves are not important depending on context as we assume its the words that carry valid information to evaluate
- Special characters, punctiations, accent marks that just aid in tone of speach have no predictive value, hence must be removed
- conversions to lowercase are important as meaning of The and the is same, yet a different value is stored for each word, hence it increases dimentionality as it would be stored as a separate word due to capital letter difference, and we actually may miss removal or search for these words as such search just by default is lowercap - it would not catch a capitalized letter word..
- removing the same word variations - particularly verbs that conjugate and carry time tense - ran,running, run, runned etc .. have really same root word, and add much dimentionality with no value to predictive power as the word really in classification means the same thing in terms of categories which is by root word

There is a quick way to address this - a stemmer that just cuts prefixes and suffixes quickly and more or less arrives at a reduced list of words, and while a few may be garbled in general it does the job well increasing speed of computation and data reduction to valid set, at much greater magnitude that some loss of proper words - his becoming he for example by tructiating 's'. This works well with large data sets...

If we prefer accuracy over time and computing power, and/or are working with a small data set, we can use the lemmenizer that will check if the trunctated word is in the dictionary, and leave only words that do make sense - non garbled set, hence at cost of processing have a better data set full of valid words...

- Tokenization - as part of data reduction, we want to take out the most common conjucting and meaning less for analysis putposes words like - he she a the... that do not carry information we are analysing for. In order to reduce these words, strings must be broken to a list of separate words, and therefore tokenized and than compared against our list of stopwords (meaningwise worthless) to be taken out. Warning some words like no or not are important as they reverese meaning, hence depending on context may have to be taken out of the standard stopwords list

Once data is clean, well we must pass it ovet to a classifier and seems Random Forest is the recommended algorithm. But classifiers accept numbers not text, hence we must first :

- join the tokens, back into a string, because vectorizers accept strings.
- use a vectorizer to change text to numbers - actually a matrix, then converted to an array to be fed to a classifier.
- Vectorizers themselves can be limited to say 1000 most frequently occurring words in corpus as another data reduction technique, and once a vocabulary of accepted words is mapped out as features (columns), they can be fitted as a numbers in an array - thats is what vectorizes do.

While a simple vectorizer just counts how many times a word appears in a document by itself, and TF-IDF vectorizer also add a valuation (by log) how many documents this word appears in, hence adjusts the value in terms of appearance in a document as well as across all documents- a more thorough technique.

Once vectorized and text is transalated to numbers, we can feed the arrays to a classifyer and check its predictive power using the usual train/test data method.

In my particular case, the results of both vectorizers were very simular, likely due to a small relatively dataset used, hence results produced were on a small sample. Also likely, because the data was well cleaned up- there is no reason why the predictive power should differ, on a good set of clean data using the same classification algorithm Random Forest/.

A shortcut we could use in predictive sentiment analysis of special built libraries for example VADER or Text Blob - which have alredy pre built list of classified words (positive/ negatve groups).

These models can also indicate strength of sentiment, an added feature not present in standard classifiers. Also, in case we lack lables, these programs have already pre made categories in which they can label our prediction. Lastly, a nice feature here is a measure of subjectivity - level of factual information applied versus opinions.