Loading data

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
nltk.download('stopwords')
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import TimeSeriesSplit
from sklearn import preprocessing
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

#Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [ ]:

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [4]:

```
# using the SQLite Table to read data.
con = sqlite3.connect('/content/drive/MyDrive/ML/database.sqlite')
#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, con)
```

In [5]:

```
# drop duplicate rows
data1 = filtered_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},keep='first')

# drop rows that do not meet the condition
data1 = data1[data1['HelpfulnessNumerator'] <= data1['HelpfulnessDenominator']]
```

In [6]:

```
# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating.
```

```
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Label'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (525814, 11)
```

Out[6]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...

In [8]:

```
# count score values
filtered_data['Score'].value_counts()
```

Out[8]:

```
5    363122
4     80655
1     52268
2     29769
Name: Score, dtype: int64
```

In [12]:

```
# Randomely select 20000 samples from each'Score' 1,2,4,5
S1 = filtered_data[filtered_data['Score'] ==1].sample(n=2500,random_state=0)
S2 = filtered_data[filtered_data['Score'] ==2].sample(n=2500,random_state=0)
S4 = filtered_data[filtered_data['Score'] ==4].sample(n=2500,random_state=0)
S5 = filtered_data[filtered_data['Score'] ==5].sample(n=2500,random_state=0)
data2 = pd.concat([S1,S2,S4,S5])
data2.shape
data2.head(5)
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
234216	254114	B0047726E0	AGZNKD9JJ0BYY	lan	0	1	1	1318377600	cancelled but sent
438705	474410	B003SBZC1U	A17A6KEW3OF239	William Fulkerson	31	35	1	1314748800	Not Natural at all!!
844	915	B000ER6YO0	AB0BXP1IKDBIA	TreGemellini	1	1	1	1278892800	Runny and odd-tasting
428370	463254	B001FA1AWG	A140GGDL7VAUTL	J. Roper	0	2	1	1298937600	How anyone could eat this is beyond me
457270	494421	B003ZVG4WY	A3B4NB57O7J6IY	beth	0	0	1	1349222400	don't bother

[1] Text Preprocessing

[1.1] Data Cleaning: Deduplication

In [15]:

```
#Sorting data according to ProductId in ascending order
final=data2.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position
```

[1.2] Stemming, stop-word removal and Lemmatization.

In [16]:

```
# find sentences containing HTML tags
import re
i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;
```

```
3
If I could rate this fly trap lower than one star, I would.  I think flies have come from miles away
just to come in and laugh at this thing.  I'd have more success taking the flies into a vat of scalding
water than getting a fly to randomly run into this box of ridiculousness.<br />WASTE OF $$!
```

In [17]:

```
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
print(stop)
print('************************************')
print(sno.stem('tasty'))
```

```
{'until', 'of', 'to', 'off', 'himself', "should've", 'yourself', 'he', 'about', 'isn', 'this', 'again',
'through', 've', 'nor', 'our', 'with', 'such', 'no', 'hasn', 'd', 'same', 'will', 'very', 'having', 'migh
tn', "didn't", 'being', 't', "needn't", 'most', 'needn', 'his', 'doing', 'mustn', 'am', 'the', 'own', 'a
ll', "hadn't", 'between', 'here', 'against', "won't", 'they', 'or', 'who', 'is', 'herself', 'you',
"you'd", 'been', 'while', 'once', 'i', 'wasn', 'her', 'my', 'was', 'ourselves', 'if', 'm', 'in', 'its',
'where', 'aren', 'do', "isn't", 'yours', 'a', 'before', 'll', 'are', "mightn't", 'by', 'don', 'at',
'were', 'did', "couldn't", 'didn', "don't", 'y', "haven't", 'and', 'them', 'shouldn', 'won', "aren't", '
because', 'up', 's', 'as', 'hers', 'myself', 'too', 'be', 'for', "she's", "you've", 'any', 'ain',
'yourselves', 'doesn', 'shan', 'does', 'how', 'your', 'on', 'it', 'itself', 'their', 'just', "weren't",
'ma', 'during', 'not', 'but', 'few', 'that', 'can', 'o', 'after', 'has', 'had', "shouldn't", 'whom', 'th
ose', 'under', "mustn't", 'other', 'which', 'what', 'why', 'weren', 'ours', 'more', 'theirs', 'above', '
from', 'these', 'further', 'down', "wouldn't", "it's", 'themselves', "hasn't", 'haven', 'she', 'when',
'couldn', 'into', 'over', "that'll", 're', 'now', "you'll", 'an', 'me', 'have', "shan't", 'there',
'should', 'hadn', 'each', 'some', 'below', 'him', 'so', "doesn't", 'out', 'then', 'we', 'only',
'wouldn', 'both', "you're", "wasn't", 'than'}
************************************
tasti
```

In [18]:

```
#Code for implementing step-by-step the checks mentioned in the pre-processing phase

if not os.path.isfile('final.sqlite'):
    i=0
    str1=' '
    final_string=[]
    all_positive_words=[] # store words from +ve reviews here
    all_negative_words=[] # store words from -ve reviews here.
    s=''
    for sent in tqdm(final['Text'].values):
        filtered_sentence=[]
        #print(sent);
        sent=cleanhtml(sent) # remove HTMl tags
        for w in sent.split():
            for cleaned_words in cleanpunc(w).split():
                if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                    if(cleaned_words.lower() not in stop):
                        s=(sno.stem(cleaned_words.lower())).encode('utf8')
                        filtered_sentence.append(s)
                        if (final['Score'].values)[i] == 'positive':
                            all_positive_words.append(s) #list of all words used to describe positive rev
                        if(final['Score'].values)[i] == 'negative':
                            all_negative_words.append(s) #list of all words used to describe negative rev
                    else:
                        continue
                else:
                    continue
        #print(filtered_sentence)
        str1 = b" ".join(filtered_sentence) #final string of cleaned words
        #print("***********************************************************************")

        final_string.append(str1)
        i+=1

    #############---- storing the data into .sqlite file ------########################
    final['CleanedText']=final_string #adding a column of CleanedText which displays the data after pre-p
    final['CleanedText']=final['CleanedText'].str.decode("utf-8")
        # store final table into an SQlLite table for future.
    conn = sqlite3.connect('final.sqlite')
    c=conn.cursor()
    conn.text_factory = str
    final.to_sql('Reviews', conn,  schema=None, if_exists='replace', \
                 index=True, index_label=None, chunksize=None, dtype=None)
    conn.close()


    with open('positive_words.pkl', 'wb') as f:
        pickle.dump(all_positive_words, f)
    with open('negitive_words.pkl', 'wb') as f:
        pickle.dump(all_negative_words, f)
```

```
100%|██████████| 10000/10000 [00:19<00:00, 510.69it/s]
```

In [19]:

```
if os.path.isfile('final.sqlite'):
    conn = sqlite3.connect('final.sqlite')
    final = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, conn)
    conn.close()
else:
    print("Please the above cell")
```

In [20]:

```
final.to_pickle("./amazon.pkl")
```

In [22]:

```
# read data from pickle file from previous stage
data = pd.read_pickle("./amazon.pkl")
data.shape
```

Out[22]:

```
(10000, 13)
```

[2] Sorting data based on time

In [ ]:

```
# Random sampling
#df = final.take(np.random.permutation(len(final))[:10000])
#df.head(2)
```

Out[ ]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Su
297763	43030	46809	B0045AW4AA	A27QMQ9WK6YPSW	Karley	0	0	1	1326931200	
224238	252654	273910	B0026GBTQA	A1989WJVG7DHBK	OhioAtty	0	0	1	1341878400	l \ Ay

In [23]:

```
df = data
df['Time'] = pd.to_datetime(df['Time'])
# Sort by time
data = df.sort_values(by='Time')

print(data.shape)
print(data['Score'].value_counts())
```

```
(10000, 13)
5    2500
4    2500
2    2500
1    2500
Name: Score, dtype: int64
```

[3] Storing into train and test

In [26]:

```
data.head(5)
```

Out[26]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
6	346041	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19	23	1	1970-01-01 00:00:00.948240000	V CL E E
7	346053	374357	B00004CI84	A31RM5QU797HPJ	Drez	1	2	4	1970-01-01 00:00:01.024531200	ba
9	346040	374342	B00004CI84	A10L8O1ZMUIMR2	G. Kleinschmidt	61	79	2	1970-01-01 00:00:01.040947200	Gr t
52	388413	419994	B0000A0BS5	A238V1XTSK9NFE	Andrew Lynn	46	59	2	1970-01-01 00:00:01.064361600	N
53	38889	42227	B0000A0BS8	A1IU7S4HCK1XK0	Joanna Daneman	5	5	4	1970-01-01 00:00:01.067644800	I g

In [27]:

```
# Spliting into Train and test
#X_train, X_test, y_train, y_test = train_test_split(data['CleanedText'].values,data['Score'].values,tes
X=data['CleanedText'].values
y_score = data['Score'].values
y_label = data['Label'].values
```

[4] Bag of Words (BoW)

In [77]:

```
X=data['CleanedText'].values
y_score = data['Score'].values
y_label = data['Label'].values
```

In [78]:

```
#Bag of words
count_vect = CountVectorizer(max_features=1000, min_df=10)

X_bow = count_vect.fit_transform(X)
#Normalize Data
X_bow = preprocessing.normalize(X_bow)
print("Train Data Size: ",X_bow.shape)
```

```
Train Data Size:  (10000, 1000)
```

In [90]:

```
from scipy.sparse import csr_matrix

df = pd.DataFrame(data=csr_matrix.todense(X_bow))
df.to_csv('data_BOW.csv', index=False)
```

[5] TF-IDF

In [53]:

```
# Spliting into Train and test
X=data['CleanedText'].values
y_score = data['Score'].values
y_label = data['Label'].values
```

In [54]:

```
tfidf = TfidfVectorizer(ngram_range=(1,2), max_features=500, min_df=10) #Using bi-grams
X_tfidf = tfidf.fit_transform(X)
#Normalize Data
X_tfidf = preprocessing.normalize(X_tfidf)
```

```
print("Train Data Size: ",X_tfidf.shape)
```

```
Train Data Size:  (10000, 500)
```

In [62]:

```
data_tfidf = pd.DataFrame(X_tfidf)
data_tfidf['Score'] = y_score
data_tfidf['Label'] = y_label
data_tfidf
```

Out[62]:

	0	Score	Label
0	(0, 475)\t0.19076536226522964\n (0, 474)\t0...	1	0
1	(0, 226)\t0.3916743389841021\n (0, 262)\t0....	4	1
2	(0, 89)\t0.11714468575883168\n (0, 455)\t0....	2	0
3	(0, 290)\t0.0423628920510666\n (0, 478)\t0....	2	0
4	(0, 442)\t0.34726816820715223\n (0, 205)\t0...	4	1
...	...	...	...
9995	(0, 282)\t0.3320056698143062\n (0, 88)\t0.4...	2	0
9996	(0, 334)\t0.12382720220294353\n (0, 123)\t0...	5	1
9997	(0, 287)\t0.3045800949770464\n (0, 494)\t0....	5	1
9998	(0, 13)\t0.4241331868249202\n (0, 121)\t0.5...	1	0
9999	(0, 228)\t0.13630237752075827\n (0, 85)\t0....	5	1

10000 rows × 3 columns

In [64]:

```
# save the dataframe as a csv file
data_tfidf.to_csv("data_tfidf.csv")
```

[6] Word2Vec

In [36]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sent=[]
for sent in data['CleanedText'].values:
    list_of_sent.append(sent.split())
```

In [37]:

```
print(data['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent[0])
```

```
alway enjoy movi funni entertain didnt hesit pick clamshel edit guess market plan make movi famili
someth elimin strong profan element usual edit televis version warn want uncut version avoid clamshel
edit
*****************************************************************
['alway', 'enjoy', 'movi', 'funni', 'entertain', 'didnt', 'hesit', 'pick', 'clamshel', 'edit', 'guess',
'market', 'plan', 'make', 'movi', 'famili', 'someth', 'elimin', 'strong', 'profan', 'element', 'usual',
'edit', 'televis', 'version', 'warn', 'want', 'uncut', 'version', 'avoid', 'clamshel', 'edit']
```

In [38]:

```
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

In [39]:

```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  4671
sample words  ['alway', 'enjoy', 'movi', 'funni', 'entertain', 'didnt', 'hesit', 'pick', 'edit',
'guess', 'market', 'plan', 'make', 'famili', 'someth', 'elimin', 'strong', 'element', 'usual',
'version', 'warn', 'want', 'avoid', 'simpli', 'kind', 'sinc', 'michael', 'play', 'titl', 'charact', 'ghos
t', 'like', 'mischief', 'call', 'coupl', 'baldwin', 'get', 'rid', 'peopl', 'live', 'hous', 'let',
'know', 'one', 'person', 'favorit', 'said', 'feel', 'need', 'tell']
```

[7] Avg Word2Vec

In [40]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 10000/10000 [00:10<00:00, 910.53it/s]
10000
50
```

In [50]:

```
# Spliting into Train and test
X_word2vec = sent_vectors
y_score = data['Score'].values
y_label = data['Label'].values
# X_train, X_test, y_train, y_test = train_test_split(sent_vectors,data['Score'].values,test_size=0.3,sh
```

In [51]:

```
data_word2vec = pd.DataFrame(X_word2vec)
data_word2vec['Score'] = y_score
data_word2vec['Label'] = y_label
data_word2vec
```

Out[51]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	
0	-0.062367	-0.010484	0.090721	-0.495039	-0.010046	-0.318024	-0.333292	-0.285306	0.476455	0.112020	0.135831	-0.192288	-0.360204	0.3
1	-0.084306	0.030614	-0.018873	-0.389342	-0.178785	-0.225033	-0.251902	-0.157756	0.280491	0.012036	0.198278	-0.132402	-0.258020	0.2
2	-0.062945	0.072410	0.026902	-0.387559	-0.118387	-0.395339	-0.265048	-0.143297	0.455327	0.129164	0.224109	-0.150651	-0.267757	0.4
3	0.262811	-0.442554	0.284031	-0.476259	-0.058890	-0.418597	-0.280494	-0.795190	0.788272	0.365903	-0.148225	-0.267626	-0.505537	0.2
4	0.112760	-0.406984	0.286712	-0.650219	0.087212	-0.430976	-0.285638	-0.802249	0.858078	0.407038	-0.184023	-0.192551	-0.603733	0.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	-0.013014	-0.180647	0.083503	-0.768636	0.110313	-0.528782	-0.371433	-0.510604	0.586986	0.366945	0.009275	0.104732	-0.329429	0.4
9996	-0.126833	-0.021447	0.102065	-0.656402	-0.109530	-0.335807	-0.384855	-0.320340	0.626076	0.185640	0.055476	-0.361432	-0.492913	0.1
9997	0.086552	-0.017139	0.484629	-0.591558	0.006040	-0.522785	-0.386826	-0.510550	0.956098	0.422709	-0.103917	-0.204290	-0.649221	0.2
9998	-0.037238	-0.012677	0.161395	-0.426446	-0.414804	-0.359584	-0.444232	-0.213422	0.515725	0.094174	0.265202	-0.189269	-0.437315	0.2
9999	-0.262840	0.155367	0.064437	-0.900232	-0.166296	-0.527844	-0.744691	-0.355308	0.701734	0.199670	0.047554	-0.243584	-0.854363	0.3

10000 rows × 52 columns

In [52]:

```
# save the dataframe as a csv file
data_word2vec.to_csv("data_word2vec.csv")
```

[8] TF-IDF Word2Vec

In [44]:

```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(data['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [45]:

```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 10000/10000 [00:15<00:00, 628.87it/s]
```

In [46]:

```
# Spliting into Train and test
X_tfidf_word2vec = tfidf_sent_vectors
y_score = data['Score'].values
y_label = data['Label'].values
#X_train, X_test, y_train, y_test = train_test_split(tfidf_sent_vectors, data['Score'].values, test_size=
```

In [47]:

```
data_tfidf_word2vec = pd.DataFrame(X_tfidf_word2vec)
data_tfidf_word2vec['Score'] = y_score
data_tfidf_word2vec['Label'] = y_label
data_tfidf_word2vec
```

Out[47]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	
0	-0.046881	-0.015105	0.092017	-0.401055	-0.090607	-0.325169	-0.297168	-0.252889	0.431492	0.117205	0.131543	-0.170710	-0.331624	0.2
1	-0.060067	0.014853	-0.000460	-0.312015	-0.175160	-0.217285	-0.240094	-0.142320	0.265984	0.014941	0.186484	-0.125922	-0.248121	0.2
2	-0.061501	0.073079	0.033006	-0.367333	-0.133849	-0.395722	-0.239342	-0.159746	0.462475	0.128700	0.181782	-0.119313	-0.253847	0.3
3	0.657267	-0.984994	0.582354	-0.502777	0.039903	-0.415720	-0.229914	-1.442688	1.223933	0.523084	-0.531325	-0.488146	-0.787226	0.0
4	0.248054	-0.533198	0.372260	-0.621143	0.119701	-0.443986	-0.274354	-0.953549	0.972687	0.422921	-0.268462	-0.237674	-0.649299	0.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	0.020909	-0.195280	0.117890	-0.686506	0.043509	-0.537159	-0.368239	-0.500929	0.594955	0.335449	0.012982	0.099554	-0.340848	0.3
9996	-0.072137	-0.062902	0.131932	-0.784530	-0.031594	-0.320527	-0.411374	-0.429918	0.767537	0.193733	-0.131228	-0.401193	-0.602517	0.0
9997	0.102276	-0.047583	0.443892	-0.589184	-0.012078	-0.517465	-0.362354	-0.514404	0.876258	0.402594	-0.085853	-0.143980	-0.628702	0.1
9998	0.006259	-0.005005	0.135583	-0.375453	-0.426059	-0.452148	-0.376979	-0.220052	0.507017	0.120308	0.264145	-0.101482	-0.380568	0.2
9999	-0.541975	0.379249	-0.034109	-1.336829	0.046721	-0.630619	-0.939171	-0.223359	0.905672	0.065244	-0.136987	-0.228961	-1.163851	0.8

10000 rows × 52 columns

In [48]:

```
# save the dataframe as a csv file
data_tfidf_word2vec.to_csv("data_tfidf_word2vec.csv")
```