In [5]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [6]:

```
import pandas as pd
import numpy as np
import sqlite3
import string
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
import re
from gensim.models import word2vec
from gensim.models import keyedvectors
import pickle
from tqdm import tqdm
import os
```

In [7]:

```
con = sqlite3.connect('E:/Varun/amazon-fine-food-reviews/database.sqlite')
filtered_data = pd.read_sql_query('''
SELECT *
FROM Reviews
WHERE Score !=3
''', con)
def partition(x):
    if x > 3:
        return "positive"
    return "negative"
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)

filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (525814, 10)
```

Out[7]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Time | Summary |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | 1 | positive | 1303862400 | Good Quality Dog Food |
| **1** | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | 0 | negative | 1346976000 | Not as Advertised |
| **2** | 3 | B000LQOCH0 | ABXLMWJIXXAIN | Natalia Corres "Natalia | 1 | 1 | positive | 1219017600 | "Delight" says it al |

| Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Time | Summary |
|---|---|---|---|---|---|---|---|---|

In [15]:

```
filtered_data.shape
```

Out[15]:

```
(525814, 10)
```

In [16]:

```
sorted_data = filtered_data.sort_values('ProductId', axis= 0, ascending= True, inplace=False, kind=
'quicsort', na_position = 'last')
```

In [17]:

```
final_data= sorted_data.drop_duplicates(subset={'UserId','ProfileName','Time','Text'}, keep='first'
,
inplace=False)
final=final_data[final_data.HelpfulnessNumerator<=final_data.HelpfulnessDenominator]
final.shape
final['Score'].value_counts()
```

Out[17]:

```
positive    307061
negative     57110
Name: Score, dtype: int64
```

In [18]:

```
#Taking a sample of 2K Positive reviews and 2K Negative reviews
p = final[final['Score'] == "positive"]
p = p[:2000]
n= final[final['Score'] == "negative"]
n = n[:2000]
p.shape
n.shape
final_data=pd.concat([p,n])
final_data.shape
```

Out[18]:

```
(4000, 10)
```

In [19]:

```
labels = final_data['Score']
```

In [20]:

```
#Preprocessing
```

In [21]:

```
import re
i=0;
for sent in final_data['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break
    i += 1
```

```
6
I set aside at least an hour each day to read to my son (3 y/o). At this point, I consider myself
a connoisseur of children's books and this is one of the best. Santa Clause put this under the tre
```

e. Since then, we've read it perpetually and he loves it.<br /><br />First, this book taught him t
he months of the year.<br /><br />Second, it's a pleasure to read. Well suited to 1.5 y/o old to 4
+.<br /><br />Very few children's books are worth owning. Most should be borrowed from the
library. This book, however, deserves a permanent spot on your shelf. Sendak's best.

In [22]:

```
import nltk
nltk.download('stopwords')
stop = set(stopwords.words('english'))
sno = nltk.stem.SnowballStemmer('english')

def cleanhtml(sentence):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext

def cleanpunc(sentence):
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return cleaned

if not os.path.isfile('sample.sqlite'):
    i=0
    str1=' '
    final_string=[]
    all_positive_words=[]
    all_negative_words=[]
    s=''
    for sent in tqdm(final_data['Text'].values):
        filtered_sentence=[]
        sent=cleanhtml(sent)
        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_data['Score'].values)[i] == 'positive':
                            all_positive_words.append(s)
                        if(final_data['Score'].values)[i] == 'negative':
                            all_negative_words.append(s)
                    else:
                        continue
                else:
                    continue
        str1 = b" ".join(filtered_sentence)

        final_string.append(str1)
        i+=1

    final_data['CleanedText']=final_string
    final_data['CleanedText']=final_data['CleanedText'].str.decode("utf-8")
    conn = sqlite3.connect('sample.sqlite')
    c=conn.cursor()
    conn.text_factory = str
    final_data.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)
```

```
[nltk_data] Error loading stopwords: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>
```

In [23]:

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

```
    print("Please the above cell")
```

In [24]:

```
#Bag of Words
```

In [25]:

```
count_vect = CountVectorizer()
final_counts = count_vect.fit_transform(final['CleanedText'].values)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4000, 9592)
the number of unique words  9592
```

In [26]:

```
final_counts.shape
```

Out[26]:

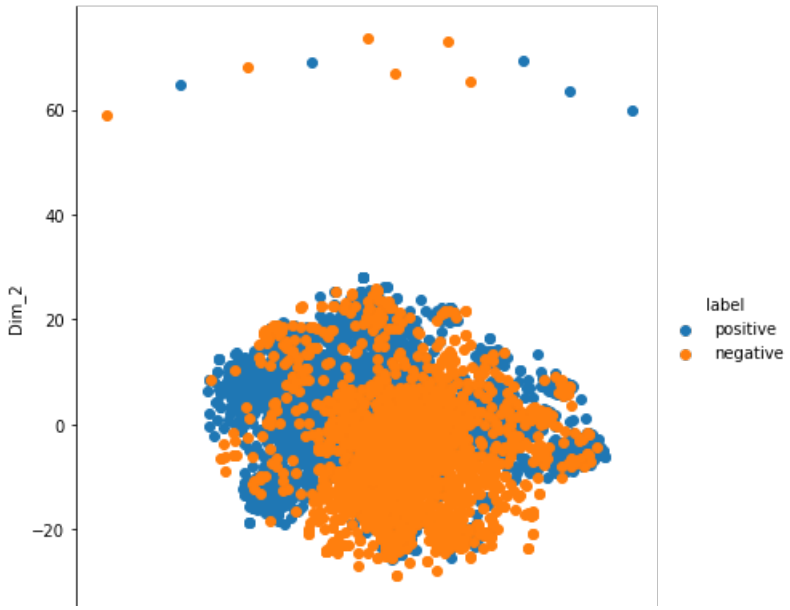
```
(4000, 9592)
```

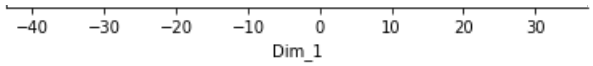
In [34]:

```
#Applying t-SNE on Bag of Words
```

In [38]:

```
from sklearn.manifold import TSNE
from scipy.sparse import csr_matrix
tsne_data= final_counts.todense() #todense() from scipy is used to covert the sparse matrix to den
se matrix.
tsne_data_labels = labels #labels= sample['Score']

model = TSNE(n_components=2, random_state=0) #Default parameters are taken i.e., perplexity = 30,
iterations= 1000
tsne_data = model.fit_transform(tsne_data)
tsne_data = np.vstack((tsne_data.T, tsne_data_labels)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()

plt.show()
```





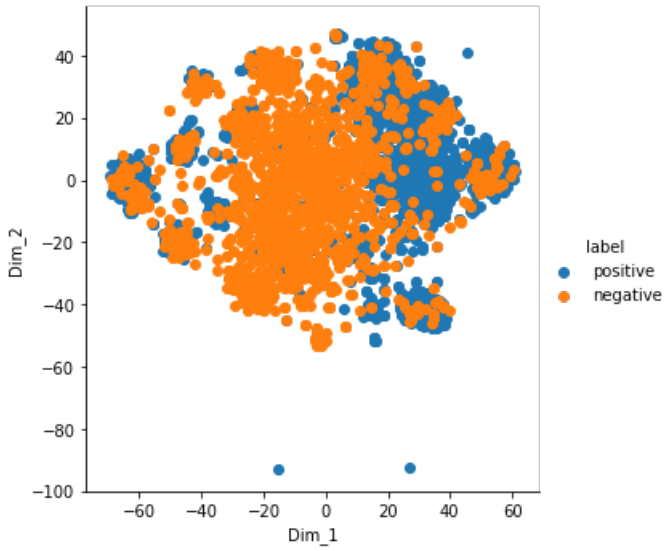

In [40]:

```
# Negative and positive reviews are overlapping by using TSNE on Bag of Words
# TF - IDF
tf_idf_vect = TfidfVectorizer()
final_tf_idf = tf_idf_vect.fit_transform(final['CleanedText'].values)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[
1
])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4000, 9592)
the number of unique words including both unigrams and bigrams  9592
```

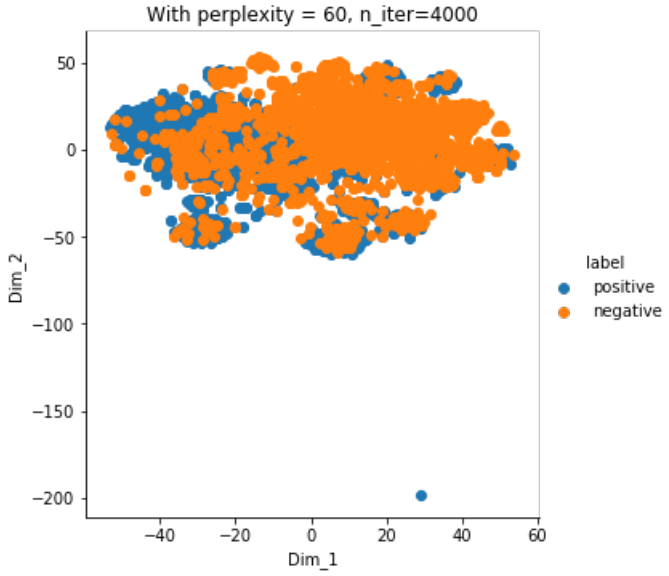
In [41]:

```
from sklearn.manifold import TSNE
from scipy.sparse import csr_matrix
tsne_data= final_tf_idf.todense() #todense() from scipy is used to covert the sparse matrix to den
se matrix.

tsne_data_labels = labels
model = TSNE(n_components=2, random_state=0) #Default parameters are taken i.e., perplexity = 30,
iterations= 1000

tsne_data = model.fit_transform(tsne_data)
tsne_data = np.vstack((tsne_data.T, tsne_data_labels)).T

tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))
sns.FacetGrid(tsne_df, hue="label", size=5).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```




In [43]:

```
# TSNE with different parameters
tsne_data= final_tf_idf.todense() #todense() from scipy is used to covert the sparse matrix to den
se matrix.
tsne_data_labels = labels
model = TSNE(n_components=2, random_state=0, perplexity=60, n_iter=4000)

tsne_data = model.fit_transform(tsne_data)
tsne_data = np.vstack((tsne_data.T, tsne_data_labels)).T

tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))
sns.FacetGrid(tsne_df, hue="label", size=5).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.title('With perplexity = 60, n_iter=4000')

plt.show()
```




In [27]:

```
#Overlapping increased very much than previous one by using different parameters
#Word2Vec
i=0
list_of_sent=[]
for sent in final['CleanedText'].values:
    list_of_sent.append(sent.split())
```

In [28]:

```
print(final['CleanedText'].values[1])
print("-------------------------------")
print(list_of_sent[1])
```

```
grew read sendak book watch realli rosi movi incorpor love son love howev miss hard cover version
paperback seem kind flimsi take two hand keep page open
-------------------------------
['grew', 'read', 'sendak', 'book', 'watch', 'realli', 'rosi', 'movi', 'incorpor', 'love', 'son', '
love', 'howev', 'miss', 'hard', 'cover', 'version', 'paperback', 'seem', 'kind', 'flimsi', 'take',
'two', 'hand', 'keep', 'page', 'open']
```

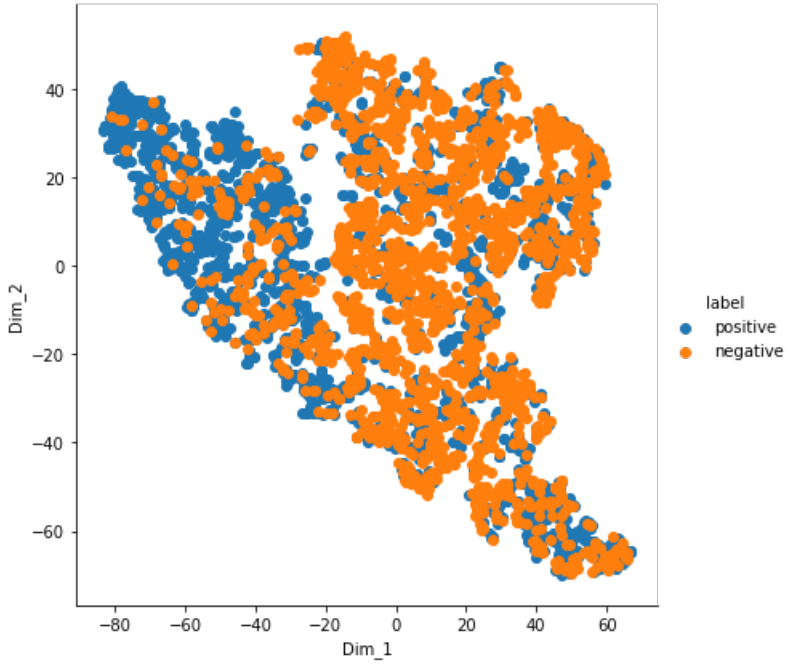
In [30]:

```
from gensim.models import Word2Vec
w2v_model= Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)

#Avg Word2Vec
sent_vectors = []
for sent in tqdm(list_of_sent):
    sent_vec = np.zeros(50)
    count_words =0
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            count_words += 1
    if count_words != 0:
        sent_vec /= count_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 4000/4000
[00:12<00:00, 326.64it/s]

4000
50
```

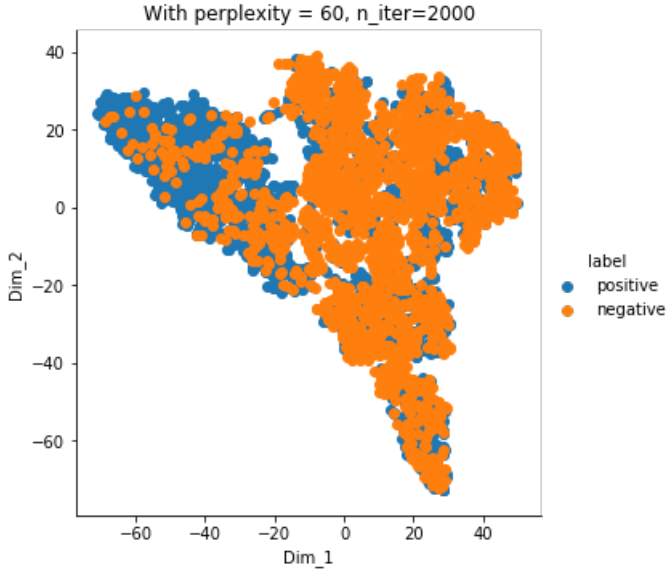
In [32]:

```
# TSNE for Avg Word2Vec
from sklearn.manifold import TSNE
w2v_data= sent_vectors
w2v_labels = labels
model = TSNE(n_components=2, random_state=0)
w2v_data = model.fit_transform(w2v_data)
w2v_data = np.vstack((w2v_data.T, w2v_labels)).T
w2v_df = pd.DataFrame(data=w2v_data, columns=("Dim_1", "Dim_2", "label"))
sns.FacetGrid(w2v_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```




In [33]:

```
# TSNE tith different parameters
w2v_data= sent_vectors
w2v_labels = labels
model = TSNE(n_components=2, random_state=0, perplexity=60, n_iter=2000)
w2v_data = model.fit_transform(w2v_data)
w2v_data = np.vstack((w2v_data.T, w2v_labels)).T
w2v_df = pd.DataFrame(data=w2v_data, columns=("Dim_1", "Dim_2", "label"))
sns.FacetGrid(w2v_df, hue="label", size=5).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.title('With perplexity = 60, n_iter=2000')
plt.show()
```



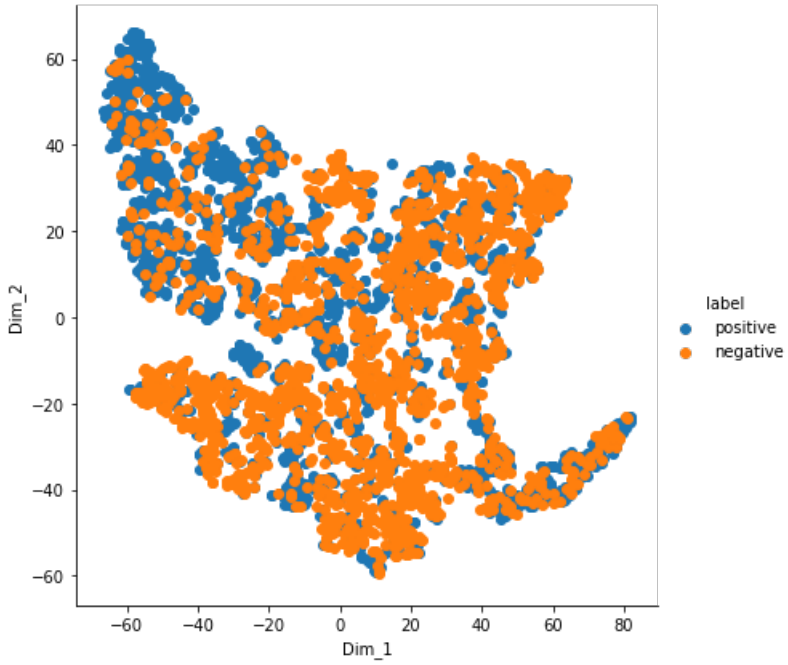
In [34]:

```
#Observation - Almost negative and positive overlapped with each other, So it is difficult to pred
ict
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(final['CleanedText'].values)
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
tfidf_feat = model.get_feature_names()
tfidf_sent_vectors = [];
row=0;

for sent in tqdm(list_of_sent):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            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%|██████████████████████████████████████████████████████████████████████████████| 4000/4000
[00:14<00:00, 285.20it/s]
```

In [35]:

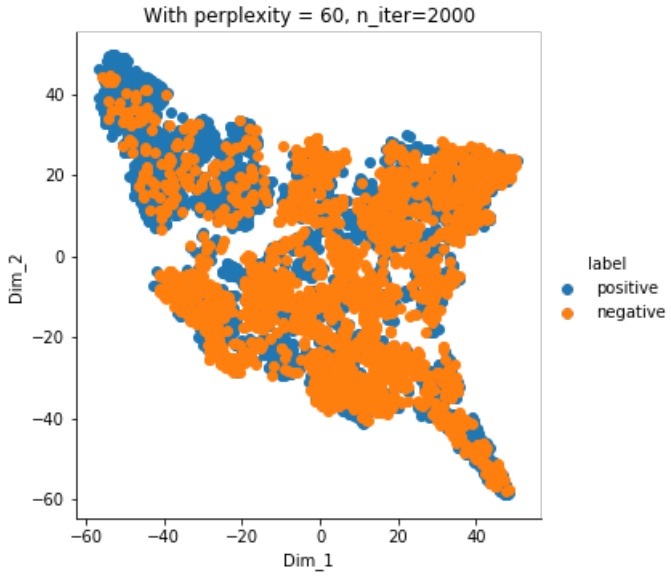
```
# TSNE on  weighted tf-idf Word2Vec
weighted_tfidf_data= tfidf_sent_vectors
weighted_tfidf_labels = labels

model = TSNE(n_components=2, random_state=0)
weighted_tfidf_data = model.fit_transform(weighted_tfidf_data)

weighted_tfidf_data = np.vstack((weighted_tfidf_data.T, weighted_tfidf_labels)).T
weighted_tfidf_df = pd.DataFrame(data=weighted_tfidf_data, columns=("Dim_1", "Dim_2", "label"))
sns.FacetGrid(weighted_tfidf_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend
()

plt.show()
```




In [36]:

```
#TSNE with different parameters
weighted_tfidf_data= tfidf_sent_vectors
```

```
weighted_tfidf_labels = labels

model = TSNE(n_components=2, random_state=0, perplexity = 60, n_iter=2000)

weighted_tfidf_data = model.fit_transform(weighted_tfidf_data)
weighted_tfidf_data = np.vstack((weighted_tfidf_data.T, weighted_tfidf_labels)).T
weighted_tfidf_df = pd.DataFrame(data=weighted_tfidf_data, columns=("Dim_1", "Dim_2", "label"))
sns.FacetGrid(weighted_tfidf_df, hue="label", size=5).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend
()

plt.title('With perplexity = 60, n_iter=2000')
plt.show()
```




In [ ]:

```
# Even using TSNE with different parameters reviews are overlapping
```