In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
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

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
```

In [3]:
```
#mounting the dataset from drive
# from google.colab import drive
# drive.mount('/content/gdrive')

#connecting to sqlite db
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data p
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LI
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3""",

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a neg
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['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[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulness
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulness
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [4]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [5]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [6]:
```
# Removing duplicate reviews
final=filtered_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}
print(final.shape)
```

```
(364173, 10)
```

In [7]:
```
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[7]: 69.25890143662969

In [8]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [9]:
```
#Before starting the next phase of preprocessing lets see the number of entries l
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(364171, 10)
```

Out[9]:
```
1    307061
0     57110
Name: Score, dtype: int64
```

In [10]:
```
final["cleanReview"] = final["Summary"].map(str) + ". " + final["Text"]
final['cleanReview'].head()
```

Out[10]:
```
0    Good Quality Dog Food. I have bought several o...
1    Not as Advertised. Product arrived labeled as ...
2    "Delight" says it all. This is a confection th...
3    Cough Medicine. If you are looking for the sec...
4    Great taffy. Great taffy at a great price.  Th...
Name: cleanReview, dtype: object
```

In [11]:
```
final['lengthOfReview'] = final['cleanReview'].str.split().str.len()
final['lengthOfReview'].head()
```

Out[11]:
```
0    52
1    34
2    98
3    43
4    29
Name: lengthOfReview, dtype: int64
```

In [10]:
```
#remove urls from text python
from tqdm import tqdm
lst = []
removed_urls_list = []
for text in tqdm(final['Text']):
  removed_urls_text = re.sub(r"http\S+", "", text)
  lst.append(removed_urls_text)
```

```
100%|██████████| 364171/364171 [00:00<00:00, 447313.57it/s]
```

In [11]:
```
#remove urls from text python
removed_urls_list = []
for text in tqdm(lst):
  removed_urls_text = re.sub(r"http\S+", "", text)
  removed_urls_list.append(removed_urls_text)
```

```
100%|██████████| 364171/364171 [00:00<00:00, 452270.97it/s]
```

In [12]:
```
from bs4 import BeautifulSoup
text_lst = []
for text in tqdm(removed_urls_list):
  soup = BeautifulSoup(text, 'lxml')
  text = soup.get_text()
  text_lst.append(text)
# print(text)
# print("="*50)
```

```
100%|██████████| 364171/364171 [01:49<00:00, 3330.00it/s]
```

In [13]:
```
print(len(final['Text']))
```

```
364171
```

In [14]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [15]:
```
decat_lst = []
for decat_text in tqdm(text_lst):
  text = decontracted(decat_text)
  decat_lst.append(text)
```

```
100%|██████████| 364171/364171 [00:05<00:00, 65510.16it/s]
```

In [16]:
```
strip_list = []
for to_strip in tqdm(decat_lst):
  text = re.sub("\S*\d\S*", "", to_strip).strip()
  strip_list.append(text)
```

```
100%|██████████| 364171/364171 [00:22<00:00, 16465.51it/s]
```

In [17]:
```
spatial_list = []
for to_spatial in tqdm(strip_list):
  text = re.sub('[^A-Za-z0-9]+', ' ', to_spatial)
  spatial_list.append(text)
```

```
100%|██████████| 364171/364171 [00:12<00:00, 29401.19it/s]
```

In [18]:
```
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'our
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itsel
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because'
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'th
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all'
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn'
            'won', "won't", 'wouldn', "wouldn't"])
```

In [19]:
```
# Combining all the above stundents
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(spatial_list):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in s
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 364171/364171 [02:44<00:00, 2216.92it/s]
```

In [20]:
```
print(len(preprocessed_reviews))
preprocessed_reviews[-1]
```

```
364171
```

Out[20]:
```
'satisfied product advertised use cereal raw vinegar general sweetner'
```

In [21]:
```
final['Preprocessed_text'] = preprocessed_reviews
```

In [22]:
```
print(len(final))
final.tail(5)
```

```
364171
```

Out[22]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	He
525809	568450	B001EO7N10	A28KG5XORO54AY	Lettie D. Carter	0	0
525810	568451	B003S1WTCU	A3I8AFVPEE8KI5	R. Sawyer	0	0
525811	568452	B004I613EE	A121AA1GQV751Z	pksd "pk_007"	2	2
525812	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	1	1
525813	568454	B001LR2CU2	A3LGQPJCZVL9UC	srfell17	0	0

In [2]:
```
dir_path = os.getcwd()
conn = sqlite3.connect(os.path.join(dir_path, 'final.sqlite'))
# final.to_sql('Reviews', conn, if_exists='replace', index=False)
```

In [3]:
```
review_3 = pd.read_sql_query(""" SELECT count(*) FROM Reviews""", conn)
print(review_3)
```

```
   count(*)
0    364171
```

In [4]:
```
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews""", conn)
```

In [5]:
```
filtered_data.shape
```

Out[5]:
```
(364171, 13)
```

In [6]:
```
filtered_data["Time"] = pd.to_datetime(filtered_data["Time"], unit = "s")
filtered_data = filtered_data.sort_values(by = "Time")
```

In [7]:
```
filtered_data.head(5)
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Hel
117924	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0
117901	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2
298792	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0
169281	230285	B00004RYGX	A344SMIA5JECGM	Vincent P. Ross	1	2
298791	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0

In [8]:
```
print(len(filtered_data))
filtered_data.info()
filtered_data = filtered_data.head(100000)
print(len(filtered_data))
```

```
364171
<class 'pandas.core.frame.DataFrame'>
Int64Index: 364171 entries, 117924 to 107253
Data columns (total 13 columns):
Id                      364171 non-null int64
ProductId               364171 non-null object
UserId                  364171 non-null object
ProfileName             364171 non-null object
HelpfulnessNumerator    364171 non-null int64
HelpfulnessDenominator  364171 non-null int64
Score                   364171 non-null int64
Time                    364171 non-null datetime64[ns]
Summary                 364171 non-null object
Text                    364171 non-null object
cleanReview             364171 non-null object
lengthOfReview          364171 non-null int64
Preprocessed_text       364171 non-null object
dtypes: datetime64[ns](1), int64(5), object(7)
memory usage: 38.9+ MB
100000
```

In [9]:
```
filtered_data['Score'].value_counts()
```

Out[9]:
```
1    87729
0    12271
Name: Score, dtype: int64
```

In [10]:
```
X = filtered_data["cleanReview"]
print(print("shape of X:", X.head(5)))
y = filtered_data["Score"]
print("shape of y:", y.head(5))
X_len = filtered_data['lengthOfReview']
```

```
shape of X: 117924    EVERY book is educational. this witty little b...
117901    This whole series is great way to spend time w...
298792    Entertainingl Funny!. Beetlejuice is a well wr...
169281    A modern day fairy tale. A twist of rumplestis...
298791    FANTASTIC!. Beetlejuice is an excellent and fu...
Name: cleanReview, dtype: object
None
shape of y: 117924    1
117901    1
298792    1
169281    1
298791    1
Name: Score, dtype: int64
```

In [11]:
```
len(filtered_data['lengthOfReview'])
```

Out[11]:
```
100000
```

In [12]:
```
X_train = X[0:40000]
Y_train = y[0:40000]
X_val = X[40000:45000]
Y_val = y[40000:45000]
X_test = X[45000:50000]
Y_test = y[45000:50000]
```

In [13]:
```
print(len(X_train), len(X_test), len(X_val))
print(len(Y_train), len(Y_test), len(Y_val))
```

```
40000 5000 5000
40000 5000 5000
```

[4.1] BAG OF WORDS

In [99]:
```
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()
X_train_vect = count_vect.fit_transform(X_train)
X_test_vect = count_vect.transform(X_test)
X_val_vect = count_vect.transform(X_val)
feature_names = count_vect.get_feature_names()
# BoW_dict = {'X_train_vect':X_train_vect, 'X_test_vect': X_test_vect, 'X_val_vec
print(X_train_vect.shape)
# print(feature_names)
```

```
(40000, 39622)
```

In [100]:
```
X_train_vect.shape
```

Out[100]:
```
(40000, 39622)
```

In [19]:
```
len(filtered_data['lengthOfReview'])
```

Out[19]:
```
100000
```

In [20]:
```
from scipy.sparse import hstack
# len_review = final['lengthOfReview'].to_sparse()
concat_data = hstack((X_train_vect,np.array(filtered_data['lengthOfReview'][0:400
concat_data_val = hstack((X_val_vect,np.array(filtered_data['lengthOfReview'][400
concat_data_test = hstack((X_test_vect,np.array(filtered_data['lengthOfReview'][4
```

In [21]:
```
print(concat_data.shape)
print(concat_data_val.shape)
print(concat_data_test.shape)
```

```
(40000, 39623)
(5000, 39623)
(5000, 39623)
```

In [22]:
```
print(len(feature_names))
```

```
39622
```

In [101]:
```
BoW_dict = {'X_train_vect':X_train_vect, 'X_test_vect': X_val_vect, 'X_val_vect':
print(BoW_dict['X_train_vect'].shape)
```

```
(40000, 39622)
```

In [102]:
```
import pickle
with open('BoW.pkl', 'wb') as handle:
    pickle.dump(BoW_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[4.3] TF-IDF

In [25]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
train_tf_idf = tf_idf_vect.fit_transform(X_train)
cv_tf_idf = tf_idf_vect.transform(X_val)
test_tf_idf = tf_idf_vect.transform(X_test)

print("the shape of out text TFIDF vectorizer ",train_tf_idf.get_shape())
print("the type of count vectorizer ",type(train_tf_idf))
print("the number of unique words including both unigrams and bigrams ", train_tf_
```

```
the shape of out text TFIDF vectorizer  (40000, 43725)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the number of unique words including both unigrams and bigrams  43725
```

In [27]:
```
tfidf_concat_data_train = hstack((train_tf_idf,np.array(filtered_data['lengthOfRe
tfidf_concat_data_val = hstack((cv_tf_idf,np.array(filtered_data['lengthOfReview'
tfidf_concat_data_test = hstack((test_tf_idf,np.array(filtered_data['lengthOfRevi
```

In [36]:
```
tf_idf_dict = {'train_tf_idf': train_tf_idf, 'cv_tf_idf': cv_tf_idf, 'test_tf_idf
```

In [37]:
```
import pickle
with open('tf_idf.pkl', 'wb') as handle:
    pickle.dump(tf_idf_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[4.4] Word2Vec

In [23]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sen=[]
for sentance in X_train:
    list_of_sen.append(sentance.split())
```

In [24]:
```
is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sen,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative3
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = Tr
```

```
[('nice', 0.9236887097358704), ('good', 0.892205536365509), ('perfect', 0.88113
4033203125), ('wonderful', 0.8648172616958618), ("It's", 0.8630595803260803),
('sauce', 0.8560157418251038), ('strong', 0.8477439284324646), ('refreshing',
0.8380926847457886), ('sweetner', 0.8370078206062317), ('very', 0.8364233970642
09)]
==================================================
[('pods.', 0.990223228931427), ('By', 0.987149715423584), ('tea..', 0.985402643
6805725), ('Pods.', 0.9850682020187378), ('best!.', 0.9844822287559509), ('clos
est', 0.9839234352111816), ('biggest', 0.9838098287582397), ('century.', 0.9837
263226509094), ('Egberts', 0.9835715293884277), ('decaf', 0.983566164970398)]
```

In [25]:
```
print(w2v_model.wv.vocab['worst'])
```

```
Vocab(count:29, index:1576, sample_int:4294967296)
```

In [116]:
```
print(len(w2v_model.wv.index2word))
```

```
12731
```

In [26]:
```
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  7578
sample words  ['Mexico', 'shop.', 'Crunchy', "/>Here's", '/>Unless', 'introduc
e', 'largest', 'next', 'dental', 'Cheese.', 'stocks', 'bother', 'BIG', 'cover',
'2004', 'it....', 'split', 'american', '2.', 'both.', 'sustainable', '"Big', 'M
akes', 'already.', 'a', 'Snack', '/>Then', 'Gift', 'favourite', 'to.', 'indiges
tion', 'nutritional', 'form', 'neither', 'happy.', 'EVERY', 'equal', '/>A', 'im
part', 'here,', 'funniest', 'step', 'PERFECT', 'pretzels', 'status', 'recover
y', 'green.', 'online,', 'willing', '2-3']
```

[4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [17]:
```
print(X_train[117924])
print(len(X_val))
print(len(X_test))
```

```
EVERY book is educational. this witty little book makes my son laugh at loud. i
recite it in the car as we're driving along and he always can sing the refrain.
he's learned about whales, India, drooping roses:  i love all the new words thi
s book  introduces and the silliness of it all.  this is a classic book i am  w
illing to bet my son will STILL be able to recite from memory when he is  in co
llege
2000
2000
```

In [18]:
```
# average Word2Vec
# compute average word2vec for each review.
def avg_w2vec(sentences_received):
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this l
    for sent in tqdm(sentences_received): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length 50, you migh
        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]))
    return sent_vectors
```

In [19]:
```
print(len([sent.split() for sent in X_train]))
```

```
5000
```

In [22]:
```
avg_w2v_train = avg_w2vec([sent.split() for sent in X_train])
```

```
100%|████████████████████████████████████████████████████████████████████████
████████████████████████████████████████| 40000/40000 [03:57<0
0:00, 168.10it/s]

40000
50
```

In [36]:
```
avg_w2v_train = avg_w2vec([sent.split() for sent in X_train])
avg_w2v_cv = avg_w2vec([sent.split() for sent in X_val])
avg_w2v_test = avg_w2vec([sent.split() for sent in X_test])
```

```
100%|██████████| 40000/40000 [06:36<00:00, 100.77it/s]

40000
50

100%|██████████| 5000/5000 [00:50<00:00, 98.37it/s]
  0%|          | 21/5000 [00:00<00:52, 95.15it/s]

5000
50

100%|██████████| 5000/5000 [00:53<00:00, 93.59it/s]

5000
50
```

In [37]:
```
Avg_w2v_dict = {'X_train_avgw2v':avg_w2v_train, 'Y_train_avgw2v': Y_train,
                      'X_val_avgw2v': avg_w2v_cv, 'Y_val_avgw2v': Y_val,
                     'X_test_avgw2v': avg_w2v_test, 'Y_test_avgw2v': Y_test}
```

In [29]:
```
Avg_w2v_dict = {'X_train_avgw2v':avg_w2v_train}
```

In [30]:
```
import pickle
with open('avg_w2v40k.pkl', 'wb') as handle:
    pickle.dump(Avg_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[4.4.1.2] TFIDF weighted W2v

In [20]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)
# 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 [21]:
```
# 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 = t

def tfidf_w2v(sentences_received):
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored i
    row=0;
    for sent in tqdm(sentences_received): # 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 and word in tfidf_feat:
                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

    return tfidf_sent_vectors
```

In [255]:
```
tfidf_w2v_train = tfidf_w2v([sent.split() for sent in X_train])
```

```
100%|██████████████████████████████████████████████████████████████████
██████████████████████████████████████████████████| 40000/40000 [17:35<0
0:00, 37.90it/s]
```

In [41]:
```
tfidf_w2v_train = tfidf_w2v([sent.split() for sent in X_train])
tfidf_w2v_cv = tfidf_w2v([sent.split() for sent in X_val])
tfidf_w2v_test = tfidf_w2v([sent.split() for sent in X_test])
```

```
  1%|          | 250/40000 [00:11<37:44, 17.55it/s]

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-41-ecba5fd8bed9> in <module>()
----> 1 tfidf_w2v_train = tfidf_w2v([sent.split() for sent in X_train])
      2 tfidf_w2v_cv = tfidf_w2v([sent.split() for sent in X_val])
      3 tfidf_w2v_test = tfidf_w2v([sent.split() for sent in X_test])

<ipython-input-40-a918f499fe71> in tfidf_w2v(sentences_received)
     11         for word in sent: # for each word in a review/sentence
     12             if word in w2v_words and word in tfidf_feat:
---> 13                 vec = w2v_model.wv[word]
     14     #            tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
     15                 # to reduce the computation we are

KeyboardInterrupt:
```

In [74]:
```
tfidf_w2v_dict = {'X_train_tfidfw2v':tfidf_w2v_train, 'Y_train_tfidfw2v': Y_train
                      'X_val_tfidfw2v': tfidf_w2v_cv, 'Y_val_tfidfw2v': Y_val,
                     'X_test_tfidfw2v': tfidf_w2v_test, 'Y_test_tfidfw2v': Y_test}
```

In [256]:
```
tfidf_w2v_dict = {'X_train_tfidfw2v':tfidf_w2v_train}
```

In [60]:
```
with open('tfidf_w2v.pkl', 'wb') as handle:
    pickle.dump(tfidf_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [257]:
```
with open('tfidf_w2v40k.pkl', 'wb') as handle:
    pickle.dump(tfidf_w2v_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

K-Means on BoW

In [103]:
```
import pickle
# with open(r"/content/gdrive/My Drive/Colab Notebooks/Assignment 4/BoW.pkl", "rb
with open(r"BoW.pkl", "rb") as input_file:
    BoW_dict = pickle.load(input_file)
```

In [25]:
```
from scipy.sparse import vstack
X_train_val = vstack((BoW_dict['X_train_vect'], BoW_dict['X_val_vect']))
```

In [26]:
```
Y_train_val = pd.concat([Y_train, Y_val], axis= 0)
```

In [27]:
```
print(X_train_val.shape)
print(Y_train_val.shape)
```

```
(45000, 39623)
(45000,)
```

In [29]:
```
from sklearn.cluster import KMeans
from sklearn.model_selection import GridSearchCV
from tqdm import tqdm
bow_k_inertia_train = dict()

for k_val in tqdm(range(1, 15)):
    bow_km_clf = KMeans(n_clusters = k_val, n_jobs = -1)
    bow_km_clf.fit(BoW_dict['X_train_vect'])
    bow_k_inertia_train[k_val] = (bow_km_clf.inertia_)
    bow_k_inertia_train['model'] = bow_km_clf
```

```
  0%|          | 0/14 [00:00<?, ?it/s]
  7%|▋         | 1/14 [00:22<04:51, 22.45s/it]
 14%|█▍        | 2/14 [02:22<10:21, 51.83s/it]
 21%|██▏       | 3/14 [04:25<13:24, 73.17s/it]
 29%|██▊       | 4/14 [09:10<22:47, 136.70s/it]
 36%|███▌      | 5/14 [13:35<26:16, 175.19s/it]
 43%|████▎     | 6/14 [17:06<24:46, 185.85s/it]
 50%|█████     | 7/14 [22:52<27:17, 233.98s/it]
 57%|█████▋    | 8/14 [26:30<22:55, 229.27s/it]
 64%|██████▍   | 9/14 [31:35<20:59, 251.94s/it]
 71%|███████▏  | 10/14 [36:27<17:35, 263.88s/it]
 79%|███████▊  | 11/14 [41:08<13:27, 269.03s/it]
 86%|████████▌ | 12/14 [44:34<08:20, 250.20s/it]
 93%|█████████▎| 13/14 [48:46<04:10, 250.50s/it]
100%|██████████| 14/14 [52:33<00:00, 243.59s/it]
```

In [30]:
```
bow_k_inertia_train
```

Out[30]:
```
{1: 58965768.423400216,
 2: 27588589.838376883,
 3: 16396440.503868137,
 'model': KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
     n_clusters=14, n_init=10, n_jobs=-1, precompute_distances='auto',
     random_state=None, tol=0.0001, verbose=0),
 5: 8305740.274215528,
 6: 6759047.977353432,
 7: 5500436.662348218,
 8: 4731157.131754027,
 9: 4220420.761793087,
 10: 3894281.1748510287,
 11: 3648164.6516318317,
 12: 3449200.497430736,
 13: 3298052.0892569493,
 14: 3164246.3862376423,
 4: 11187789.781576395}
```

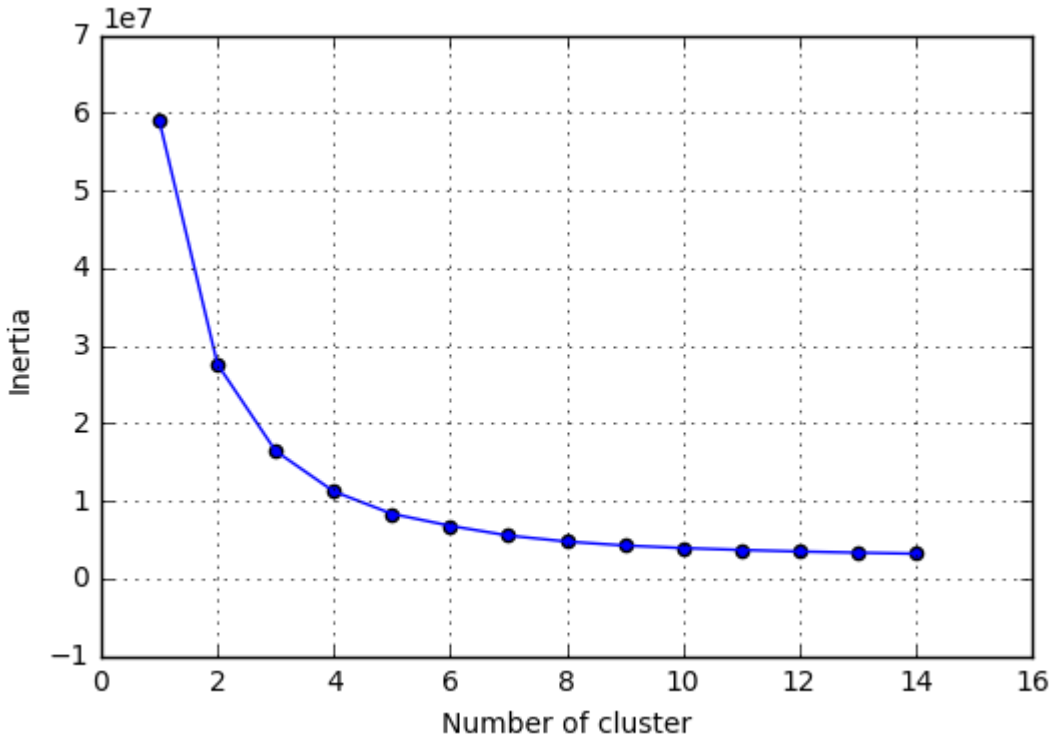
In [31]:
```
with open('bow_dict_of_clusters.pkl', 'wb') as handle:
    pickle.dump(bow_k_inertia_train, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [19]:
```
with open('bow_dict_of_clusters.pkl', 'rb') as fp:
    bow_dict_of_clusters = pickle.load(fp)
```

In [33]:
```
cluster_dict = {1: 58965768.423400216,
 2: 27588589.838376883,
 3: 16396440.503868137,
 5: 8305740.274215528,
 6: 6759047.977353432,
 7: 5500436.662348218,
 8: 4731157.131754027,
 9: 4220420.761793087,
 10: 3894281.1748510287,
 11: 3648164.6516318317,
 12: 3449200.497430736,
 13: 3298052.0892569493,
 14: 3164246.3862376423,
 4: 11187789.781576395}
```

In [34]:
```
with open('cluster_dict.pkl', 'wb') as handle:
    pickle.dump(cluster_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [47]:
```
plt.figure()
plt.plot(list(cluster_dict.keys()), list(cluster_dict.values()))
plt.scatter(list(cluster_dict.keys()), list(cluster_dict.values()))
plt.xlabel("Number of cluster")
plt.ylabel("Inertia")
# for i in range(1, 15):
#     plt.scatter(centroids[0], centroids[1], s=200, c='g', marker='s')
#     plt.scatter(2.01559419, 2.02597093, s=200, c='r', marker='s')
plt.grid()
plt.show()
```



In [48]:
```
centroids=bow_km_clf.cluster_centers_
centroids
```

Out[48]:
```
array([[1.19932838e-04, 0.00000000e+00, 1.19932838e-04, ...,
        0.00000000e+00, 0.00000000e+00, 2.03692732e+01],
       [9.19963201e-04, 9.19963201e-04, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 1.06322907e+02],
       [0.00000000e+00, 3.51246927e-04, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 6.40273973e+01],
       ...,
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        0.00000000e+00, 2.38549618e-04, 4.94730439e+01],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 1.37943730e+02],
       [1.91828122e-04, 0.00000000e+00, 0.00000000e+00, ...,
        1.91828122e-04, 0.00000000e+00, 3.81317859e+01]])
```

In [105]:
```
BoW_dict['X_train_vect'].shape
```

Out[105]:
```
(40000, 39622)
```

In [106]:
```
bow_final_clf = KMeans(n_clusters = 6, n_jobs = -1)
bow_final_clf.fit(BoW_dict['X_train_vect'])
```

Out[106]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=6, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [130]:
```
from wordcloud import WordCloud
imp_features = []

print("Top terms per cluster:")
order_centroids = bow_final_clf.cluster_centers_.argsort()[:, ::-1]
terms = count_vect.get_feature_names()
# print(len(terms))
for i in range(6):
    for ind in order_centroids[i, :10]:
        imp_featues_dict = {}
        imp_featues_dict[i] = terms[ind]
        imp_features.append(imp_featues_dict)
```

```
Top terms per cluster:
Cluster 0:
Cluster 1:
Cluster 2:
Cluster 3:
Cluster 4:
Cluster 5:
```

In [164]:
```
cl1 = [d[0] for d in imp_features if 0 in d]
cl2 = [d[1] for d in imp_features if 1 in d]
cl3 = [d[2] for d in imp_features if 2 in d]
cl4 = [d[3] for d in imp_features if 3 in d]
cl5 = [d[4] for d in imp_features if 4 in d]
cl6 = [d[5] for d in imp_features if 5 in d]
print(cl1, cl2, cl3, cl4, cl5, cl6)
```

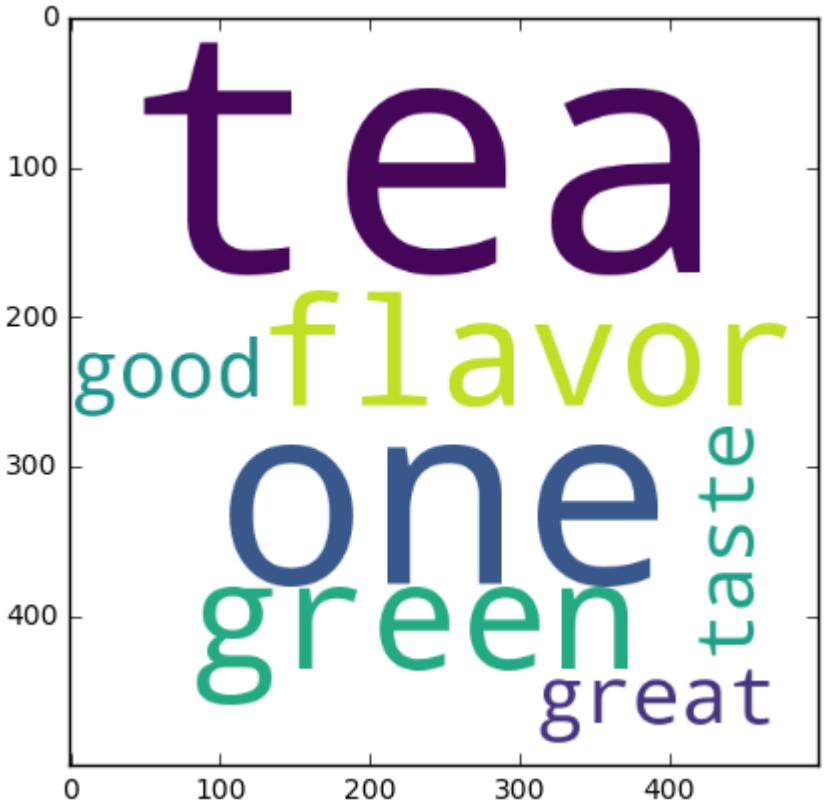
```
['tea', 'not', 'like', 'flavor', 'great', 'good', 'green', 'teas', 'taste', 'on
e'] ['not', 'like', 'one', 'would', 'good', 'taste', 'food', 'product', 'no',
'flavor'] ['coffee', 'not', 'cup', 'like', 'good', 'great', 'one', 'flavor', 'b
est', 'taste'] ['great', 'good', 'not', 'product', 'best', 'love', 'like', 'tas
te', 'one', 'flavor'] ['not', 'like', 'good', 'taste', 'one', 'great', 'would',
'product', 'flavor', 'get'] ['tea', 'not', 'teas', 'green', 'organic', 'like',
'black', 'tazo', 'good', 'taste']
```

In [165]:
```
cl1_string = ' '.join(cl1)
cl2_string = ' '.join(cl2)
cl3_string = ' '.join(cl3)
cl4_string = ' '.join(cl4)
cl5_string = ' '.join(cl5)
cl6_string = ' '.join(cl6)
```

In [166]:
```
print(cl1_string)
```

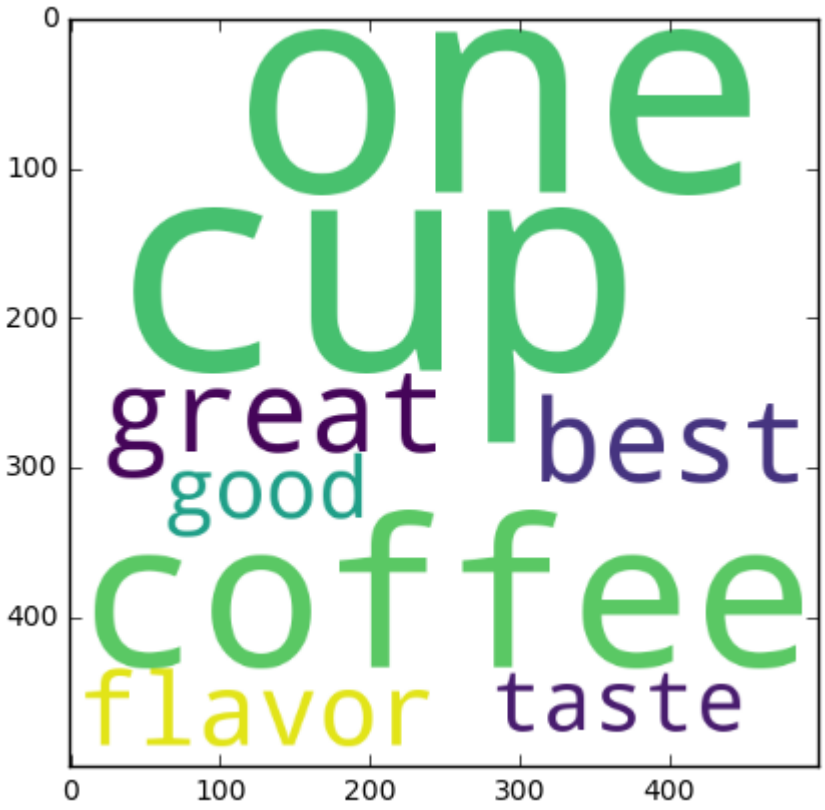
```
tea not like flavor great good green teas taste one
```

In [177]:
```
from wordcloud import WordCloud

wordcloud_cluster1 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster2 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster3 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster4 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster5 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster6 = WordCloud(width = 500, height = 500, background_color ='whit

# plot the WordCloud image
plt.imshow(wordcloud_cluster1)
plt.tight_layout(pad = 0)
plt.show()
```



In [178]:
```
plt.imshow(wordcloud_cluster2)
plt.tight_layout(pad = 0)
plt.show()
```

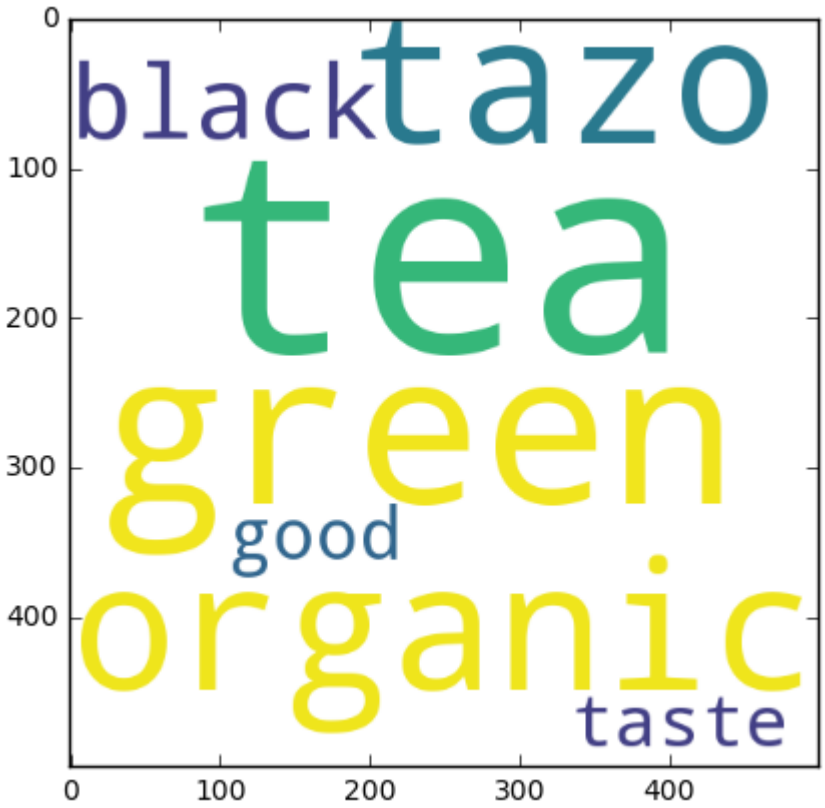
In [179]:
```
plt.imshow(wordcloud_cluster3)
plt.tight_layout(pad = 0)
plt.show()
```



In [180]:
```
plt.imshow(wordcloud_cluster4)
plt.tight_layout(pad = 0)
plt.show()
```

In [181]:
```
plt.imshow(wordcloud_cluster5)
plt.tight_layout(pad = 0)
plt.show()
```

In [182]:
```
plt.imshow(wordcloud_cluster6)
plt.tight_layout(pad = 0)
plt.show()
```



K-Means on tfidf

In [2]:
```
import pickle
# with open(r"/content/gdrive/My Drive/Colab Notebooks/Assignment 4/BoW.pkl", "rb
with open(r"tf_idf.pkl", "rb") as input_file:
    tfidf_dict = pickle.load(input_file)
```

In [3]:
```
print(tfidf_dict['train_tf_idf'].shape)
```

```
(40000, 43725)
```

In [16]:
```
from sklearn.cluster import KMeans
from sklearn.model_selection import GridSearchCV
from tqdm import tqdm
tfidf_k_inertia_train = dict()

for k_val in tqdm(range(1, 15)):
    tfidf_km_clf = KMeans(n_clusters = k_val, n_jobs = -1)
    tfidf_km_clf.fit(tfidf_dict['train_tf_idf'])
    tfidf_k_inertia_train[k_val] = (tfidf_km_clf.inertia_)
```

```
100%|██████████| 14/14 [5:44:08<00:00, 1446.30s/it]
```

In [17]:
```
tfidf_k_inertia_train
```

Out[17]:
```
{1: 213134056.3744063,
 2: 96728409.42598969,
 3: 55146609.075549215,
 4: 34599513.56567659,
 5: 23741540.32964884,
 6: 16898665.503877133,
 7: 12345721.302620674,
 8: 9436335.40372812,
 9: 7494236.050792929,
 10: 6206419.802226965,
 11: 5149919.206434929,
 12: 4405075.319938568,
 13: 3774623.910880657,
 14: 3351990.603868018}
```

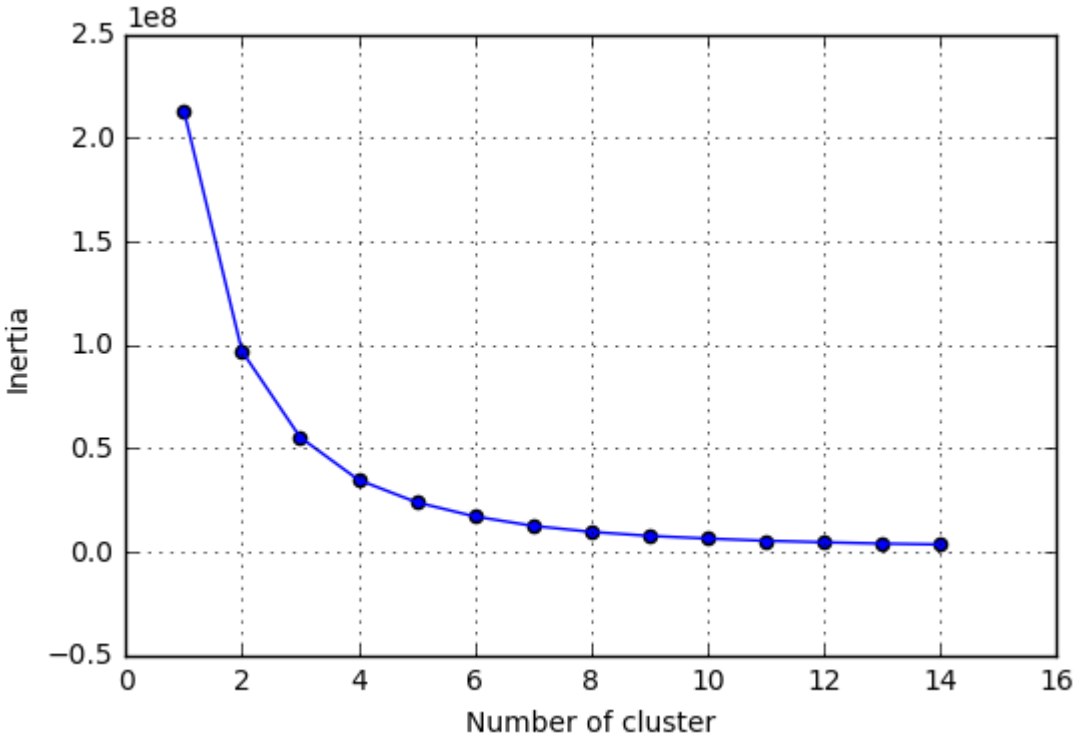
In [19]:
```
with open('tfidf_dict_of_clusters.pkl', 'wb') as handle:
    pickle.dump(tfidf_k_inertia_train, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [4]:
```
with open('tfidf_dict_of_clusters.pkl', 'rb') as fp:
    tfidf_dict_of_clusters = pickle.load(fp)
```

In [5]:
```
tfidf_dict_of_clusters
```

Out[5]:
```
{1: 213134056.3744063,
 2: 96728409.42598969,
 3: 55146609.075549215,
 4: 34599513.56567659,
 5: 23741540.32964884,
 6: 16898665.503877133,
 7: 12345721.302620674,
 8: 9436335.40372812,
 9: 7494236.050792929,
 10: 6206419.802226965,
 11: 5149919.206434929,
 12: 4405075.319938568,
 13: 3774623.910880657,
 14: 3351990.603868018}
```

In [6]:
```
plt.figure()
plt.plot(list(tfidf_dict_of_clusters.keys()), list(tfidf_dict_of_clusters.values(
plt.scatter(list(tfidf_dict_of_clusters.keys()), list(tfidf_dict_of_clusters.valu
plt.xlabel("Number of cluster")
plt.ylabel("Inertia")
plt.grid()
plt.show()
```



In [7]:
```
tfidf_dict['train_tf_idf'].shape
```

Out[7]:
```
(40000, 43725)
```

In [10]:
```
import datetime
from sklearn.cluster import KMeans
t1 = datetime.datetime.now()
tfidf_final_clf = KMeans(n_clusters = 8, n_jobs = -1)
tfidf_final_clf.fit(tfidf_dict['train_tf_idf'])
print("time required = ",datetime.datetime.now() - t1 )
```

```
time required =  1:13:50.408407
```

In [26]:
```
from wordcloud import WordCloud
imp_features_tfidf = []

print("Top terms per cluster:")
order_centroids = tfidf_final_clf.cluster_centers_.argsort()[:, ::-1]
terms = tf_idf_vect.get_feature_names()
print(len(terms))
print(tfidf_dict['train_tf_idf'].shape)
for i in range(6):
    for ind in order_centroids[i, :10]:
        imp_featues_dict = {}
        imp_featues_dict[i] = terms[ind-1]
        imp_features_tfidf.append(imp_featues_dict)
```

```
Top terms per cluster:
43725
(40000, 43725)
```

In [27]:
```
imp_features_tfidf
```

Out[27]:
```
[{0: 'coffe'},
 {0: 'thawed'},
 {0: 'cumin'},
 {0: 'irresistible'},
 {0: 'issues with'},
 {0: 'ancient'},
 {0: 'thirty'},
 {0: 'odors'},
 {0: 'this coconut'},
 {0: 'tm'},
 {1: 'issues with'},
 {1: 'thawed'},
 {1: 'irresistible'},
 {1: 'ancient'},
 {1: 'tm'},
 {1: 'thirty'},
 {1: 'odors'},
 {1: 'yorkshire tea'},
 {1: 'impulse'},
 {1: 'football'},
 {2: 'thermos'},
 {2: 'arabica'},
 {2: 'these yummy'},
 {2: 'thawed'},
 {2: 'theirs'},
 {2: 'ancient'},
 {2: 'they always'},
 {2: 'tm'},
 {2: 'odors'},
 {2: 'these and'},
 {3: 'te'},
 {3: 'thawed'},
 {3: 'issues with'},
 {3: 'this tasty'},
 {3: 'irresistible'},
 {3: 'ancient'},
 {3: 'thirty'},
 {3: 'odors'},
 {3: 'bpa'},
 {3: 'tm'},
 {4: 'bpa'},
 {4: 'br bought'},
 {4: 'thawed'},
 {4: 'ancient'},
 {4: 'tm'},
 {4: 'odors'},
 {4: 'issues with'},
 {4: 'irresistible'},
 {4: 'yorkshire tea'},
 {4: 'impulse'},
 {5: 'doesnt'},
 {5: 'thawed'},
 {5: 'hazlenut'},
 {5: 'shavings'},
 {5: 'mutt'},
```

```
 {5: 'ancient'},
 {5: 'treatments'},
 {5: 'tm'},
 {5: 'doggy'},
 {5: 'henry'}]
```

In [28]:
```
cl1 = [d[0] for d in imp_features_tfidf if 0 in d]
cl2 = [d[1] for d in imp_features_tfidf if 1 in d]
cl3 = [d[2] for d in imp_features_tfidf if 2 in d]
cl4 = [d[3] for d in imp_features_tfidf if 3 in d]
cl5 = [d[4] for d in imp_features_tfidf if 4 in d]
cl6 = [d[5] for d in imp_features_tfidf if 5 in d]
print(cl1, cl2, cl3, cl4, cl5, cl6)
```

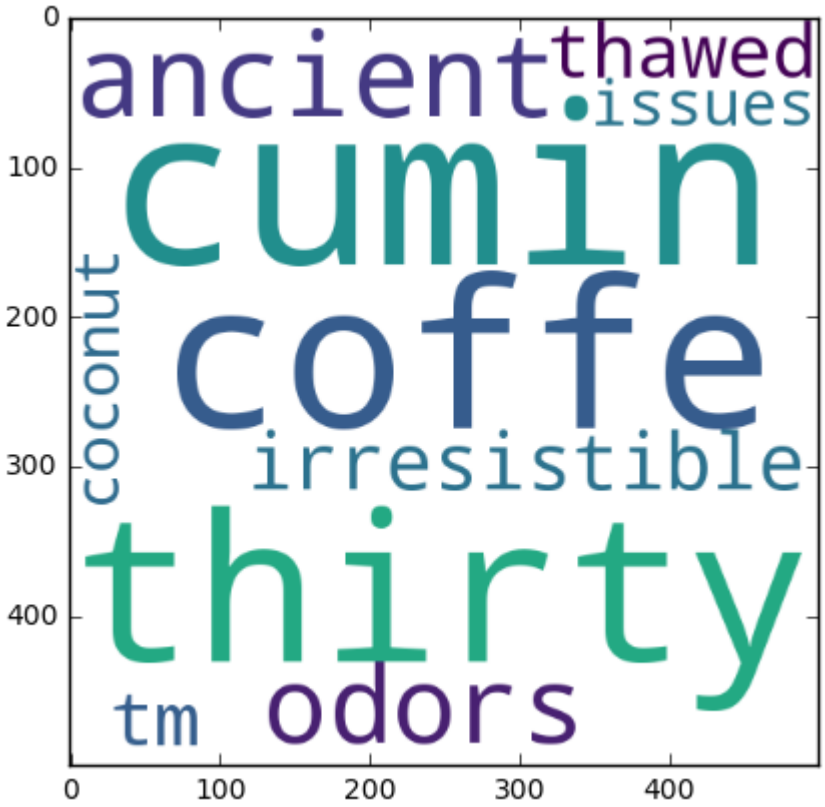
```
['coffe', 'thawed', 'cumin', 'irresistible', 'issues with', 'ancient', 'thirt
y', 'odors', 'this coconut', 'tm'] ['issues with', 'thawed', 'irresistible', 'a
ncient', 'tm', 'thirty', 'odors', 'yorkshire tea', 'impulse', 'football'] ['the
rmos', 'arabica', 'these yummy', 'thawed', 'theirs', 'ancient', 'they always',
'tm', 'odors', 'these and'] ['te', 'thawed', 'issues with', 'this tasty', 'irre
sistible', 'ancient', 'thirty', 'odors', 'bpa', 'tm'] ['bpa', 'br bought', 'tha
wed', 'ancient', 'tm', 'odors', 'issues with', 'irresistible', 'yorkshire tea',
'impulse'] ['doesnt', 'thawed', 'hazlenut', 'shavings', 'mutt', 'ancient', 'tre
atments', 'tm', 'doggy', 'henry']
```

In [29]:
```
cl1_string = ' '.join(cl1)
cl2_string = ' '.join(cl2)
cl3_string = ' '.join(cl3)
cl4_string = ' '.join(cl4)
cl5_string = ' '.join(cl5)
cl6_string = ' '.join(cl6)
```

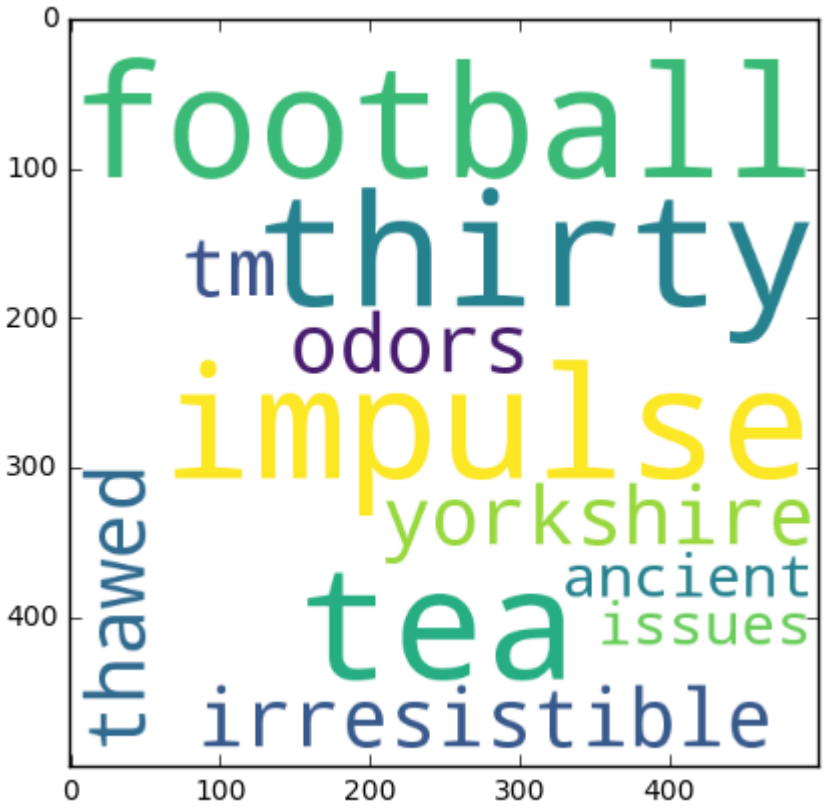
In [31]:
```
print(cl1_string)
```

```
coffe thawed cumin irresistible issues with ancient thirty odors this coconut t
m
```

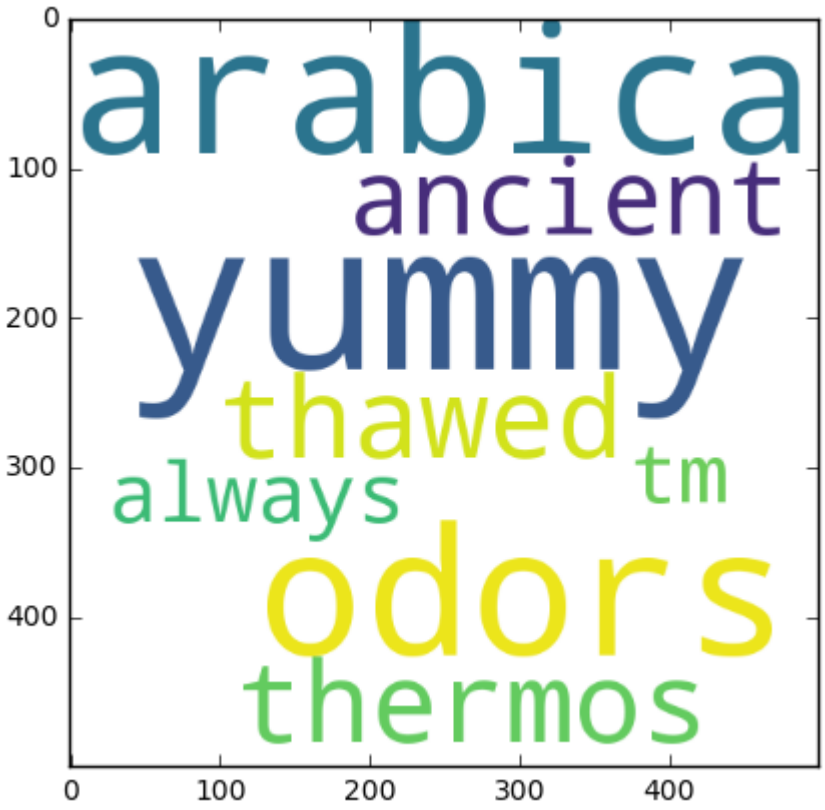
In [30]:
```
from wordcloud import WordCloud

wordcloud_cluster1 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster2 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster3 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster4 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster5 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster6 = WordCloud(width = 500, height = 500, background_color ='whit

# plot the WordCloud image
plt.imshow(wordcloud_cluster1)
plt.tight_layout(pad = 0)
plt.show()
```



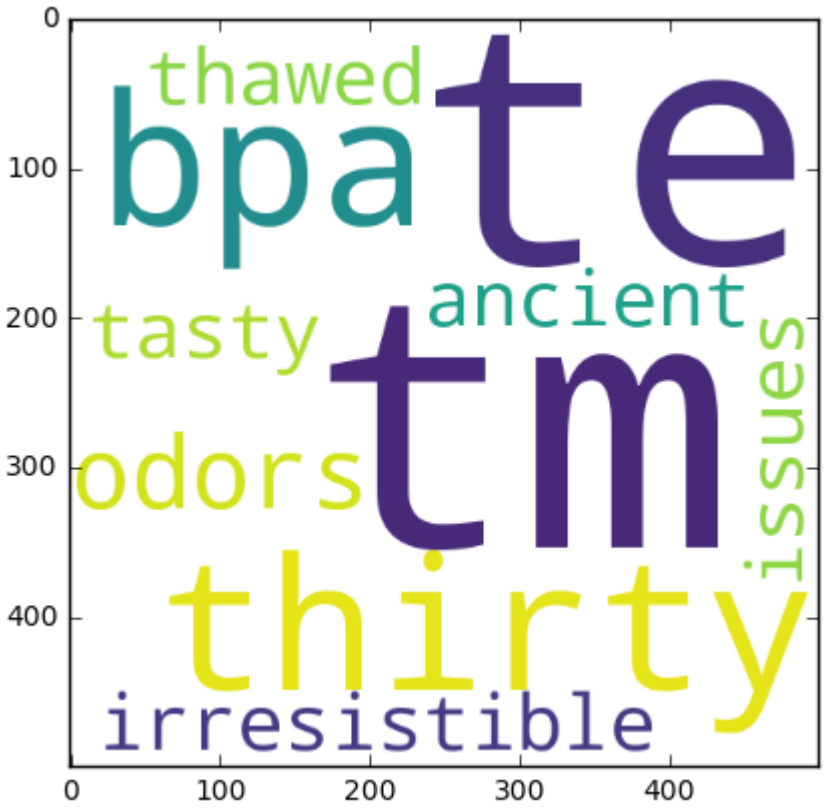
In [32]:
```
plt.imshow(wordcloud_cluster2)
plt.tight_layout(pad = 0)
plt.show()
```



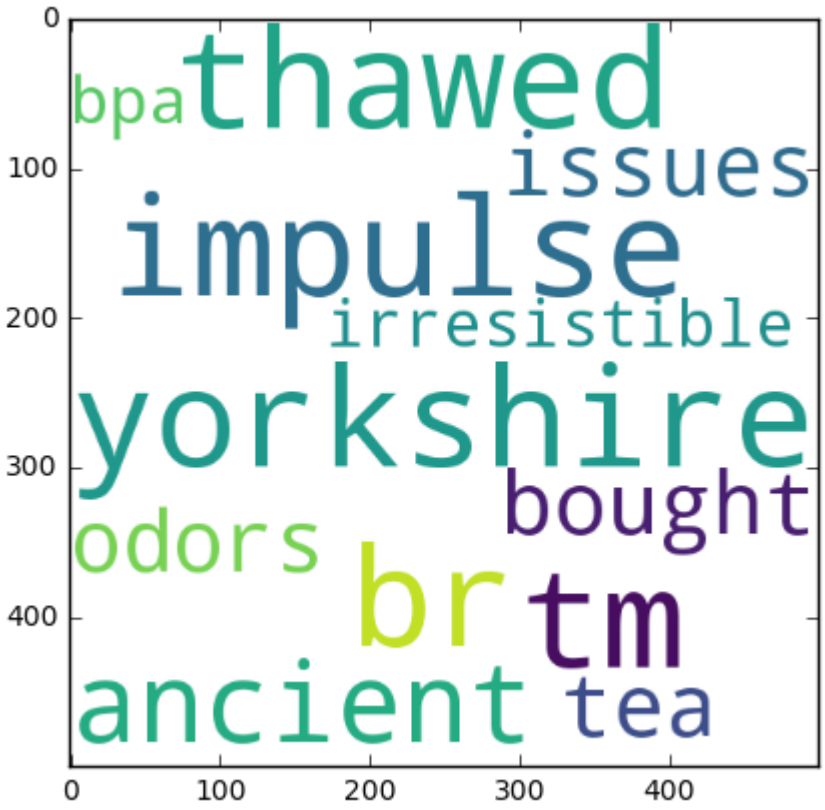
In [33]:
```
plt.imshow(wordcloud_cluster3)
plt.tight_layout(pad = 0)
plt.show()
```



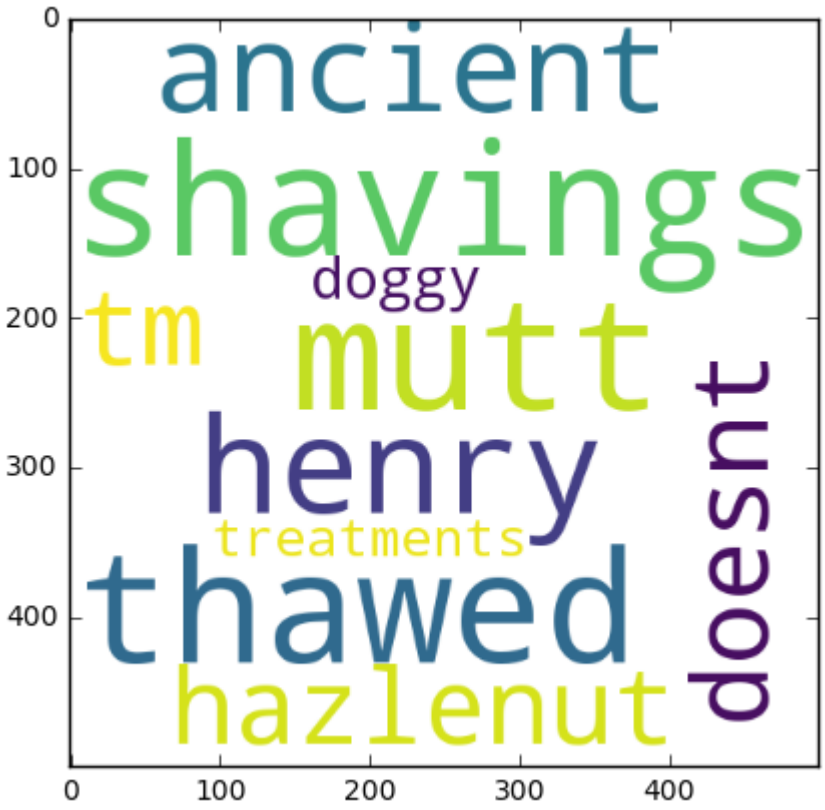
In [34]:
```
plt.imshow(wordcloud_cluster4)
plt.tight_layout(pad = 0)
plt.show()
```



In [35]:
```
plt.imshow(wordcloud_cluster5)
plt.tight_layout(pad = 0)
plt.show()
```



In [36]:
```
plt.imshow(wordcloud_cluster6)
plt.tight_layout(pad = 0)
plt.show()
```



K-Means on Avg-w2v

In [33]:
```
import pickle
# with open(r"/content/gdrive/My Drive/Colab Notebooks/Assignment 4/BoW.pkl", "rb
with open(r"avg_w2v40k.pkl", "rb") as input_file:
    avg_w2v_dict = pickle.load(input_file)
```

In [34]:
```
print(len(avg_w2v_dict['X_train_avgw2v']))
```

```
40000
```

In [82]:
```
from sklearn.cluster import KMeans
from sklearn.model_selection import GridSearchCV
from tqdm import tqdm
avg_k_inertia_train = dict()

# for k_val in tqdm([5, 10, 15, 20, 25, 30, 35, 40, 45, 50]):
for k_val in tqdm(range(1, 15)):
    avg_km_clf = KMeans(n_clusters = k_val, n_jobs = -1)
    avg_km_clf.fit(avg_w2v_dict['X_train_avgw2v'])
    avg_k_inertia_train[k_val] = (avg_km_clf.inertia_)
```

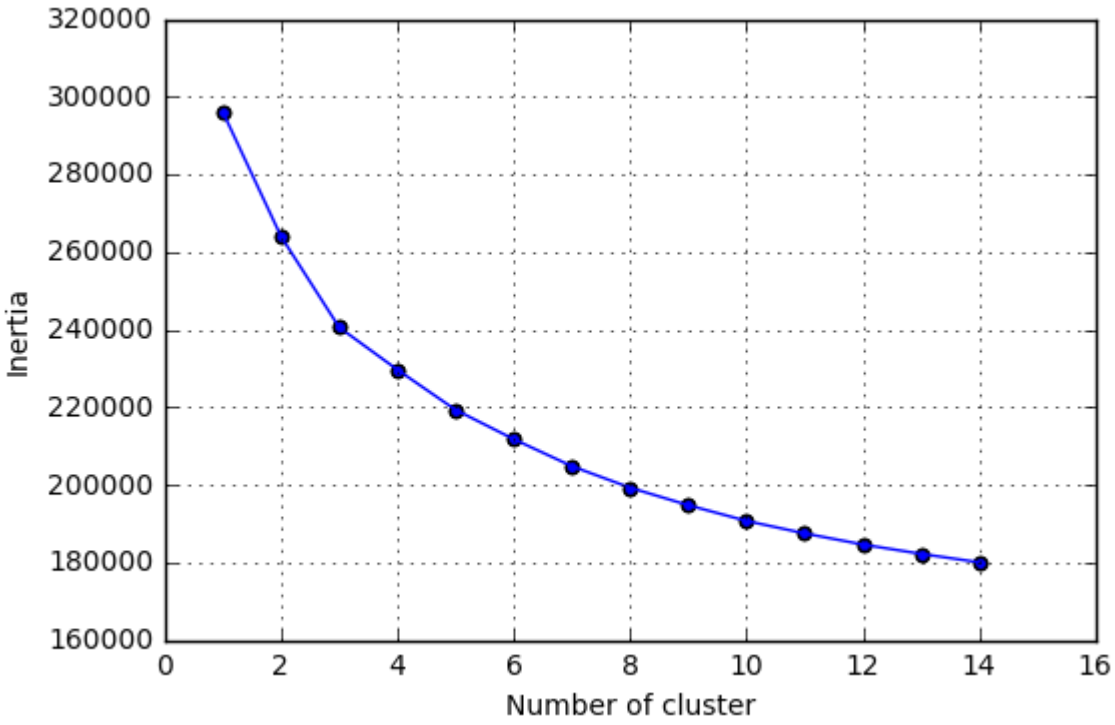
```
100%|██████████████████████████████████████████████████████████████| 14/14 [00:53<00:00,  3.85s/it]
```

In [91]:
```
complete_dict
```

Out[91]:
```
{0: array([  347,   519,   718, ..., 39973, 39976, 39978], dtype=int64),
 1: array([    1,     4,    38, ..., 39934, 39940, 39951], dtype=int64),
 2: array([   57,   101,   105, ..., 39968, 39975, 39979], dtype=int64),
 3: array([   15,    71,    79, ..., 39956, 39969, 39989], dtype=int64),
 4: array([  111,   130,   133, ..., 39963, 39995, 39996], dtype=int64),
 5: array([  145,   219,   556, ..., 39899, 39902, 39946], dtype=int64),
 6: array([   28,    40,    48, ..., 39862, 39900, 39972], dtype=int64),
 7: array([   98,   136,   165, ..., 39957, 39970, 39983], dtype=int64),
 8: array([   14,    77,    87, ..., 39960, 39977, 39981], dtype=int64),
 9: array([   69,   192,   256, ..., 39988, 39993, 39994], dtype=int64),
 10: array([   85,    90,    92, ..., 39980, 39992, 39999], dtype=int64),
 11: array([   74,   112,   121, ..., 39927, 39958, 39964], dtype=int64),
 12: array([    0,     2,     3, ..., 39984, 39986, 39997], dtype=int64),
 13: array([  261,   321,   326, ..., 39990, 39991, 39998], dtype=int64)}
```

In [ ]:
```
import collections
ordered_avg = collections.OrderedDict(sorted(avg_k_inertia_train.items()))
ordered_avg
```

In [92]:
```
plt.figure()
plt.plot(list(ordered_avg.keys()), list(ordered_avg.values()))
plt.scatter(list(ordered_avg.keys()), list(ordered_avg.values()))
# plt.plot(list([5, 10, 15, 20, 25, 30, 35, 40, 45, 50]), (219282.18364842678, 19
plt.scatter(list(avg_k_inertia_train.keys()), list(avg_k_inertia_train.values()))
plt.xlabel("Number of cluster")
plt.ylabel("Inertia")
plt.grid()
plt.show()
```



In [93]:
```
best_avg_km_clf = KMeans(n_clusters = 8, n_jobs = -1)
best_avg_km_clf.fit(avg_w2v_dict['X_train_avgw2v'])
```

Out[93]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=8, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [94]:
```
#Labels of each point
best_avg_km_clf.labels_
mydict = {i: np.where(best_avg_km_clf.labels_ == i)[0] for i in range(best_avg_km_
# Transform this dictionary into list (if you need a list as result)
dict_list = []
key_list = []
value_list = []
for key, value in mydict.items():
    key_list.append(key)
    value_list.append(value)
complete_dict = dict(zip(key_list, value_list))
```

In [198]:
```
for index, values in complete_dict.items():
    print(index, len(values), values)
```

```
0 4570 [   67   130   133 ... 39995 39996 39998]
1 5476 [  136   219   315 ... 39976 39977 39978]
2 8435 [    0     1     2 ... 39984 39986 39997]
3 6221 [   15    64    71 ... 39969 39989 39999]
4 6760 [   14    74    77 ... 39981 39983 39992]
5 3448 [   57   101   105 ... 39968 39975 39979]
6 2438 [   28    40    48 ... 39862 39900 39972]
7 2652 [   69   192   256 ... 39988 39993 39994]
```

In [149]:
```
all_dict = {}
for index, values in complete_dict.items():
    review_list = []
    for individual_review in values:
        review_list.append(X_train.iloc[individual_review])
    all_dict[index] = review_list
```

Out[149]:
```
{0: ['almost expired sept returned due short brief expiration date sept would
less use pods amazon pull shelves',
  'great gift holidays brother law thrilled gift italian superfast shipping',
  'excellent quality service received gift great quality time delivery bunch
clubs wine beer fruit flowers coffee cigars chocolate pizza cool idea main we
b site www clubsofamerica com',
  'live plants amazon sure not purchased plants via catalog years frustrating
make call find first three choices not stock worse place order find upon deli
very certain items stock like purchasing amazon know whether not inventory pa
rticular item make purchase not pulled trigger one bonsai yet giving stars ef
fort next amazon livestock',
  'everything excellent highly recommend business company like job nice price
quick shipping tasty product',
  'service smile purchased organic fruit basket gift coworker although item n
ot excellent condition first delivered company replaced no questions asked ku
dos lydia even adding bonus gift compensate recipient replacement product exc
ellent quality',
  'outstanding product service wife dark chocolate snobs scharffenberger orga
nic trader joe dark favorites went low carb began hunting alternatives everyt
hing tried sorely lacking either odd taste bad texture causing serious stomac
```

In [214]:
```
# [x for xs in lst for x in xs.split(',')]
words_dict = {}
for key in all_dict.keys():
    r = " ".join(all_dict[key])
    words_dict[key] = r
```

In [220]:
```
final_dict = {}
for other_key in words_dict.keys():
    g = words_dict[other_key].split(' ')
    final_dict[other_key] = g
```

In [237]:
```
from collections import Counter
cluster_dict = {}
for k in final_dict.keys():
    top_words = []
    c = Counter(final_dict[k])
    for i,j in c.most_common(20):
        top_words.append(i)
    cluster_dict[k] = top_words
```

In [241]:
```
cluster_dict[0]
```

Out[241]:
```
['not',
 'amazon',
 'great',
 'product',
 'price',
 'good',
 'find',
 'shipping',
 'order',
 'buy',
 'store',
 'best',
 'get',
 'would',
 'grocery',
 'love',
 'stores',
 'local',
 'excellent',
 'time']
```

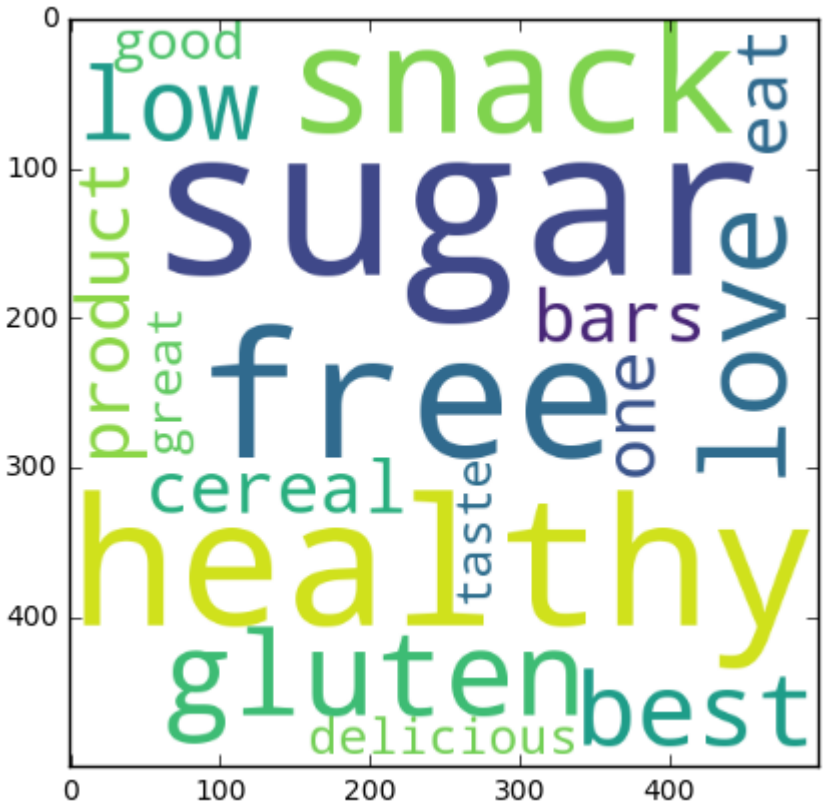
In [53]:
```
with open('avg_dict_of_clusters.pkl', 'wb') as handle:
    pickle.dump(avg_k_inertia_train, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [4]:
```
with open('tfidf_dict_of_clusters.pkl', 'rb') as fp:
    tfidf_dict_of_clusters = pickle.load(fp)
```

In [242]:
```
cl1_string = ' '.join(cluster_dict[0])
cl2_string = ' '.join(cluster_dict[1])
cl3_string = ' '.join(cluster_dict[2])
cl4_string = ' '.join(cluster_dict[3])
cl5_string = ' '.join(cluster_dict[4])
cl6_string = ' '.join(cluster_dict[5])
cl7_string = ' '.join(cluster_dict[6])
cl8_string = ' '.join(cluster_dict[7])
```

In [243]:
```
from wordcloud import WordCloud

wordcloud_cluster1 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster2 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster3 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster4 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster5 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster6 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster7 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster8 = WordCloud(width = 500, height = 500, background_color ='whit


# plot the WordCloud image
plt.imshow(wordcloud_cluster1)
plt.tight_layout(pad = 0)
plt.show()
```

In [244]:
```
plt.imshow(wordcloud_cluster2)
plt.tight_layout(pad = 0)
plt.show()
```

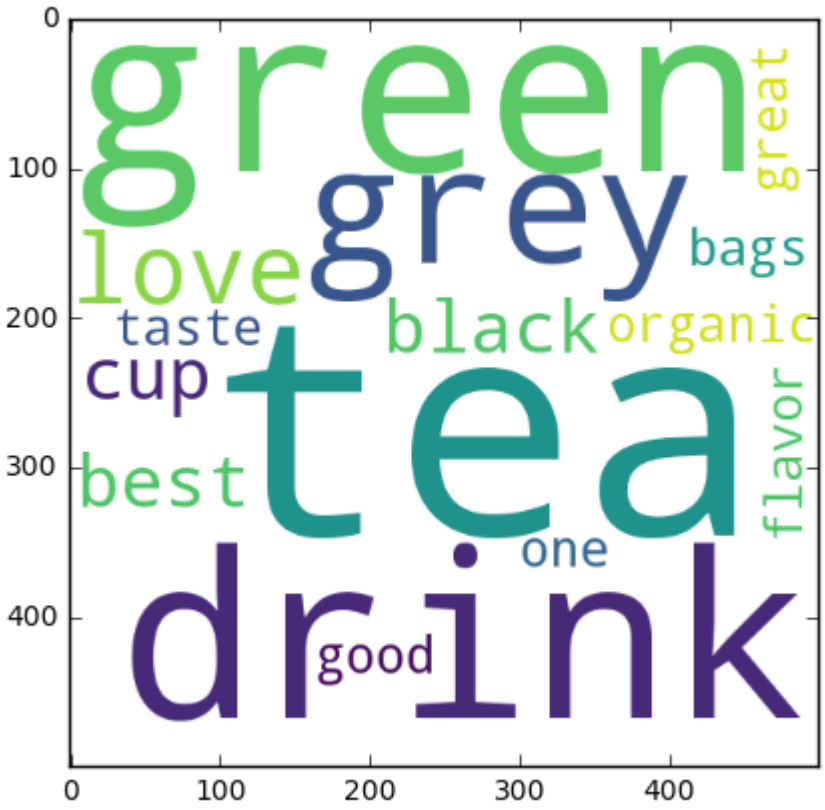


In [245]:
```
plt.imshow(wordcloud_cluster3)
plt.tight_layout(pad = 0)
plt.show()
```

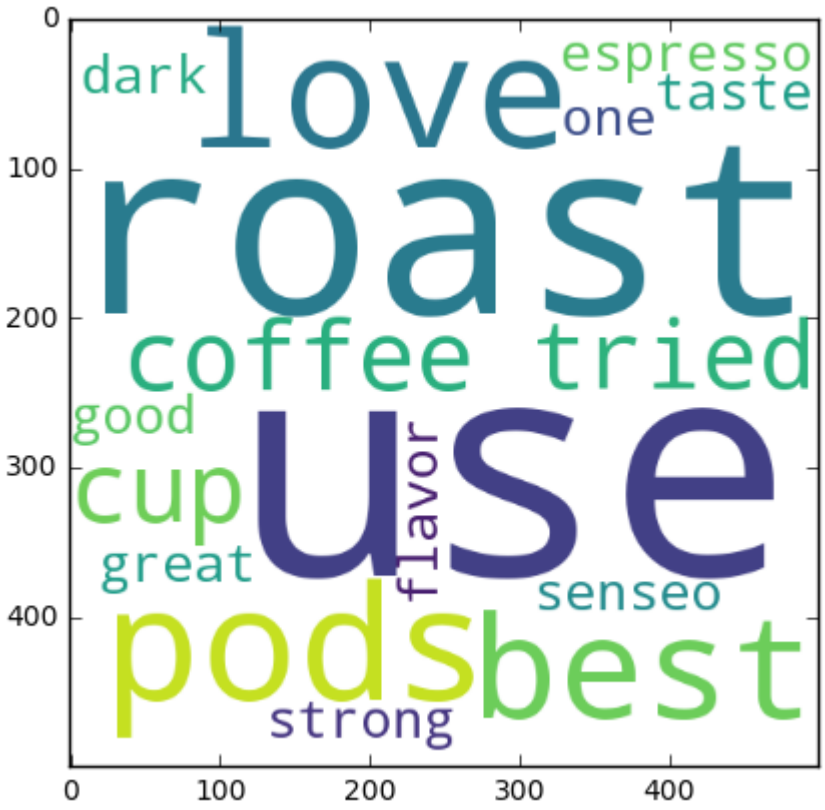
In [246]:
```
plt.imshow(wordcloud_cluster4)
plt.tight_layout(pad = 0)
plt.show()
```

In [247]:
```
plt.imshow(wordcloud_cluster5)
plt.tight_layout(pad = 0)
plt.show()
```

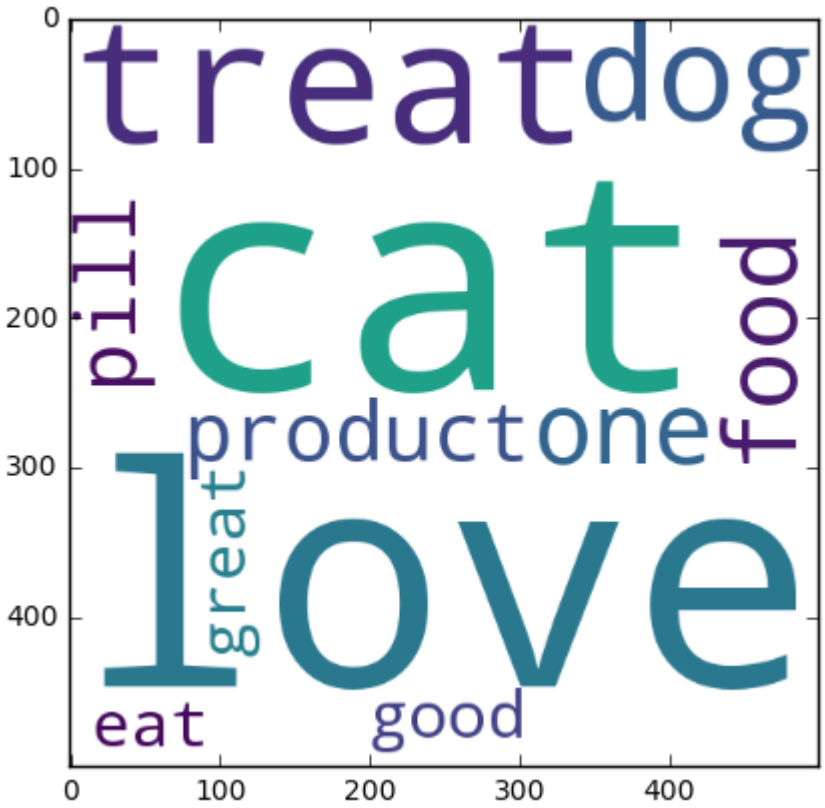
In [248]:
```
plt.imshow(wordcloud_cluster6)
plt.tight_layout(pad = 0)
plt.show()
```



In [249]:
```
plt.imshow(wordcloud_cluster7)
plt.tight_layout(pad = 0)
plt.show()
```



In [250]:
```
plt.imshow(wordcloud_cluster8)
plt.tight_layout(pad = 0)
plt.show()
```



K-Means on Tfidf-w2v

In [258]:
```
import pickle
# with open(r"/content/gdrive/My Drive/Colab Notebooks/Assignment 4/BoW.pkl", "rb
with open(r"tfidf_w2v40k.pkl", "rb") as input_file:
    tfidf_w2v_dict = pickle.load(input_file)
```

In [259]:
```
print(len(tfidf_w2v_dict['X_train_tfidfw2v']))
```

```
40000
```

In [260]:
```
from sklearn.cluster import KMeans
from sklearn.model_selection import GridSearchCV
from tqdm import tqdm
tfidfw2v_k_inertia_train = dict()

for k_val in tqdm(range(1, 15)):
    tfw2v_km_clf = KMeans(n_clusters = k_val, n_jobs = -1)
    tfw2v_km_clf.fit(tfidf_w2v_dict['X_train_tfidfw2v'])
    tfidfw2v_k_inertia_train[k_val] = (tfw2v_km_clf.inertia_)
```

```
100%|██████████| 14/14 [01:20<00:00,  5.72s/it]
```

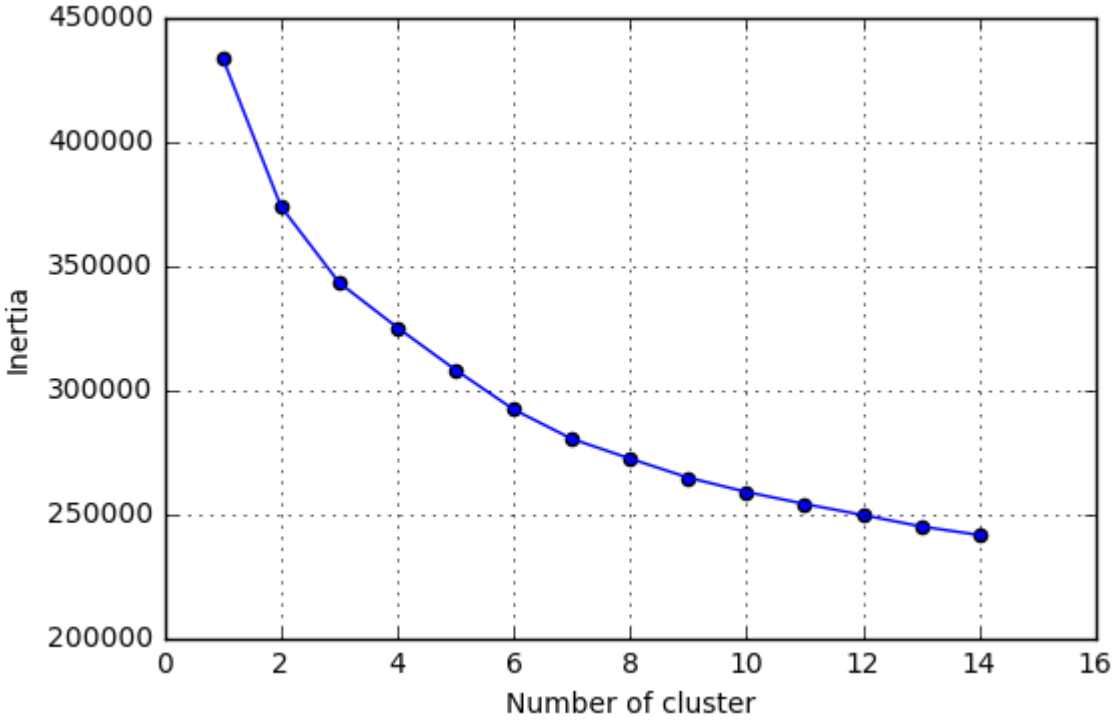
In [261]:
```
tfidfw2v_k_inertia_train
```

Out[261]:
```
{1: 433370.35119516205,
 2: 373657.0616496744,
 3: 343035.4823576481,
 4: 324948.03423364135,
 5: 307911.3179946581,
 6: 291956.9950212743,
 7: 280124.2655523777,
 8: 272174.43128803174,
 9: 264510.7237561073,
 10: 258761.74303204715,
 11: 253915.02057746705,
 12: 249476.01910216716,
 13: 244877.42827188625,
 14: 241423.33251849338}
```

In [263]:
```
with open('tfidfw2v_dict_of_clusters.pkl', 'wb') as handle:
    pickle.dump(tfidfw2v_k_inertia_train, handle, protocol=pickle.HIGHEST_PROTOCO
```

In [264]:
```
with open('tfidfw2v_dict_of_clusters.pkl', 'rb') as fp:
    tfidfw2v_dict_of_clusters = pickle.load(fp)
```

In [262]:
```
plt.figure()
plt.plot(list(tfidfw2v_k_inertia_train.keys()), list(tfidfw2v_k_inertia_train.val
plt.scatter(list(tfidfw2v_k_inertia_train.keys()), list(tfidfw2v_k_inertia_train.
plt.xlabel("Number of cluster")
plt.ylabel("Inertia")
plt.grid()
plt.show()
```



In [320]:
```
best_tfw2v_km_clf = KMeans(n_clusters = 8, n_jobs = -1)
best_tfw2v_km_clf.fit(tfidf_w2v_dict['X_train_tfidfw2v'])
```

Out[320]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=8, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [321]:
```
#Labels of each point
best_tfw2v_km_clf.labels_
mydict = {i: np.where(best_tfw2v_km_clf.labels_ == i)[0] for i in range(best_tfw2
# Transform this dictionary into list (if you need a list as result)
dict_list = []
key_list = []
value_list = []
for key, value in mydict.items():
    key_list.append(key)
    value_list.append(value)
complete_dict = dict(zip(key_list, value_list))
```

In [322]:
```
complete_dict
```

Out[322]:
```
{0: array([   67,    91,   111, ..., 39995, 39996, 39998], dtype=int64),
 1: array([   71,    79,    81, ..., 39981, 39989, 39997], dtype=int64),
 2: array([   57,   101,   105, ..., 39968, 39975, 39979], dtype=int64),
 3: array([   74,    77,    92, ..., 39980, 39992, 39999], dtype=int64),
 4: array([  136,   193,   233, ..., 39976, 39977, 39978], dtype=int64),
 5: array([    0,     1,     2, ..., 39983, 39984, 39986], dtype=int64),
 6: array([   28,    40,    48, ..., 39874, 39940, 39972], dtype=int64),
 7: array([   69,   192,   256, ..., 39988, 39993, 39994], dtype=int64)}
```

In [323]:
```
for index, values in complete_dict.items():
    print(index, len(values), values)
```

```
0 4940 [   67    91   111 ... 39995 39996 39998]
1 6218 [   71    79    81 ... 39981 39989 39997]
2 2755 [   57   101   105 ... 39968 39975 39979]
3 5478 [   74    77    92 ... 39980 39992 39999]
4 5786 [  136   193   233 ... 39976 39977 39978]
5 10478 [    0     1     2 ... 39983 39984 39986]
6 2120 [   28    40    48 ... 39874 39940 39972]
7 2225 [   69   192   256 ... 39988 39993 39994]
```

In [327]:
```
all_dict = {}
for index, values in complete_dict.items():
    review_list = []
    for individual_review in values:
        review_list.append(X_train.iloc[individual_review])
    all_dict[index] = review_list
```

In [328]:
```
# [x for xs in lst for x in xs.split(',')]
words_dict = {}
for key in all_dict.keys():
    r = " ".join(all_dict[key])
    words_dict[key] = r
```

In [329]:
```
final_dict = {}
for other_key in words_dict.keys():
    g = words_dict[other_key].split(' ')
    final_dict[other_key] = g
```

In [330]:
```
from collections import Counter
cluster_dict = {}
for k in final_dict.keys():
    top_words = []
    c = Counter(final_dict[k])
    for i,j in c.most_common(50):
        top_words.append(i)
    cluster_dict[k] = top_words
```

In [331]:
```
cluster_dict[0]
```

Out[331]:
```
['not',
 'great',
 'product',
 'amazon',
 'price',
 'good',
 'shipping',
 'order',
 'find',
 'would',
 'buy',
 'get',
 'store',
 'one',
 'ordered',
 'best',
 'time',
 'love',
 'box',
 'gift',
 'no',
 'excellent',
 'service',
 'like',
 'stores',
 'arrived',
 'grocery',
 'item',
 'local',
 'tea',
 'received',
 'much',
 'found',
 'taste',
 'quality',
 'well',
 'bought',
 'candy',
 'could',
 'company',
 'fast',
 'free',
 'really',
 'fresh',
 'better',
 'bag',
 'got',
 'coffee',
 'bags',
 'even']
```

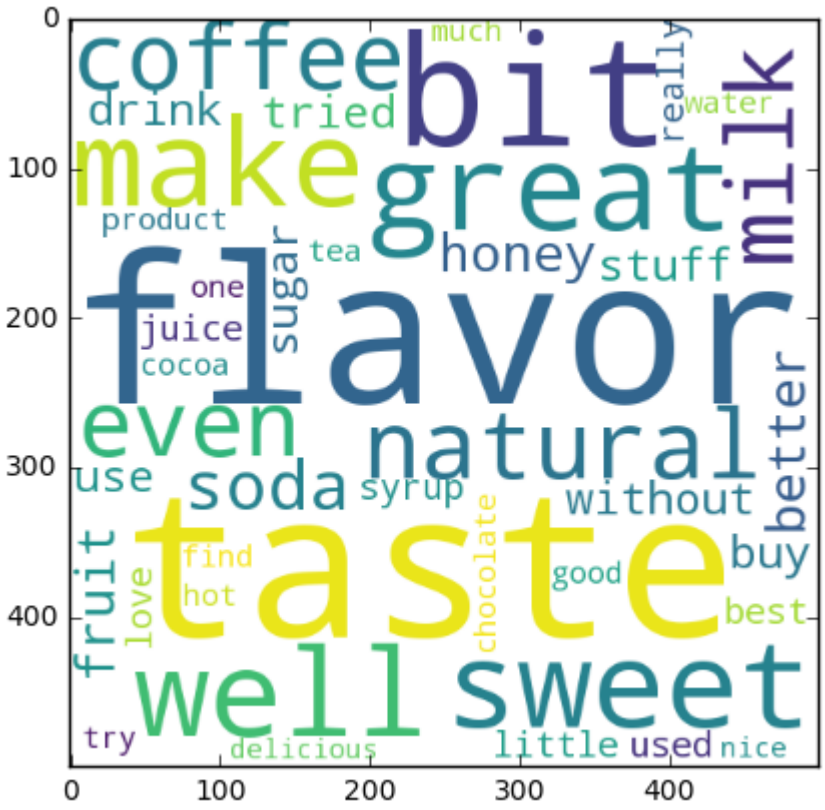
In [333]:
```
cl1_string = ' '.join(cluster_dict[0])
cl2_string = ' '.join(cluster_dict[1])
cl3_string = ' '.join(cluster_dict[2])
cl4_string = ' '.join(cluster_dict[3])
cl5_string = ' '.join(cluster_dict[4])
cl6_string = ' '.join(cluster_dict[5])
cl7_string = ' '.join(cluster_dict[6])
cl8_string = ' '.join(cluster_dict[7])
```

In [334]:
```
from wordcloud import WordCloud

wordcloud_cluster1 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster2 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster3 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster4 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster5 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster6 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster7 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster8 = WordCloud(width = 500, height = 500, background_color ='whit


# plot the WordCloud image
plt.imshow(wordcloud_cluster1)
plt.tight_layout(pad = 0)
plt.show()
```

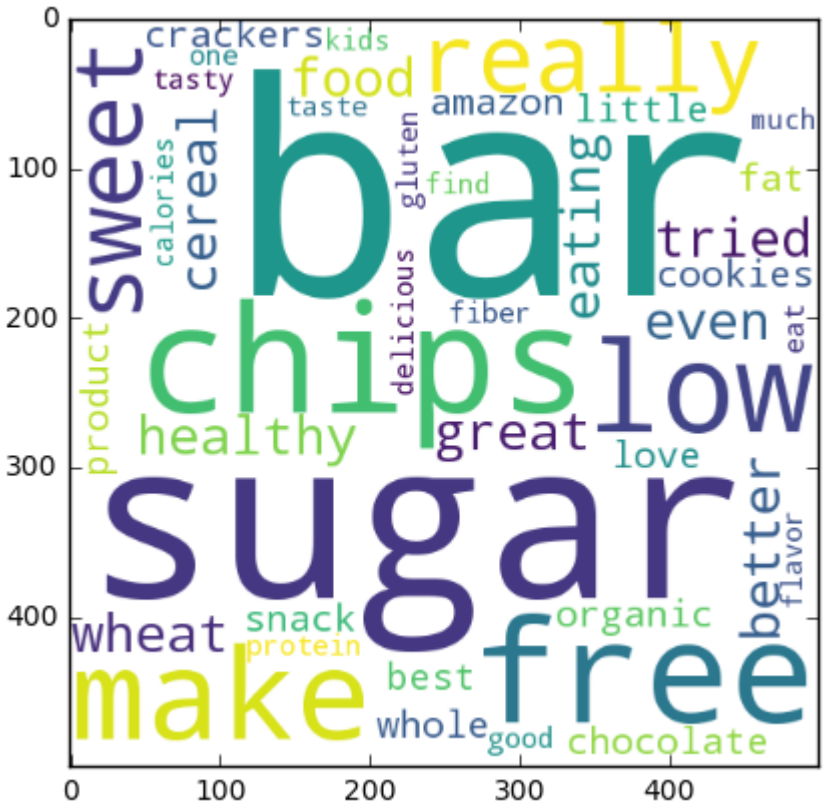
In [335]:
```
plt.imshow(wordcloud_cluster2)
plt.tight_layout(pad = 0)
plt.show()
```

In [336]:
```
plt.imshow(wordcloud_cluster3)
plt.tight_layout(pad = 0)
plt.show()
```

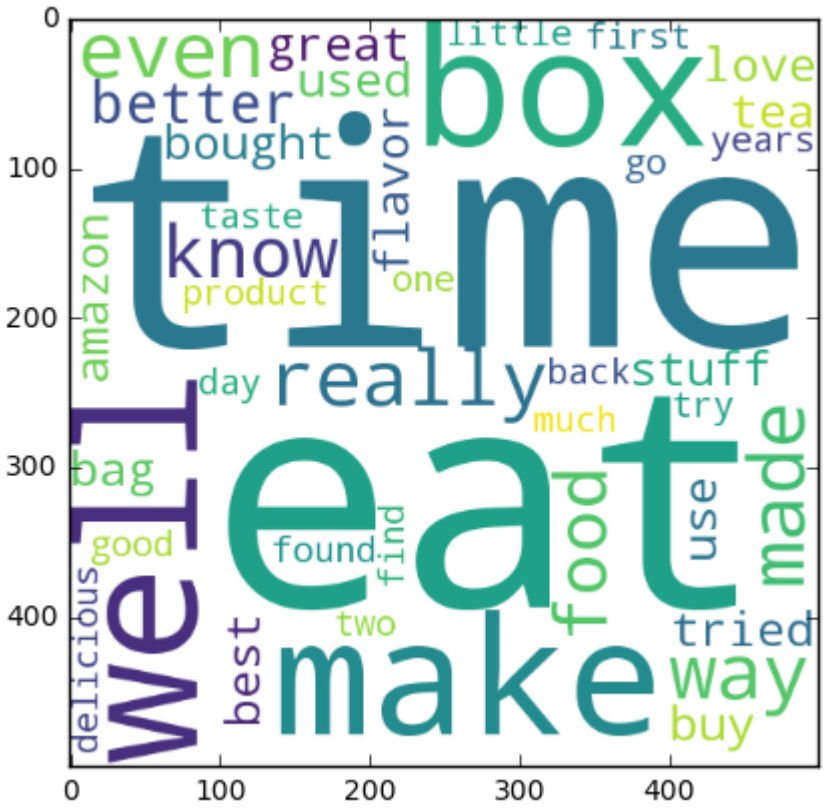
In [337]:
```
plt.imshow(wordcloud_cluster4)
plt.tight_layout(pad = 0)
plt.show()
```



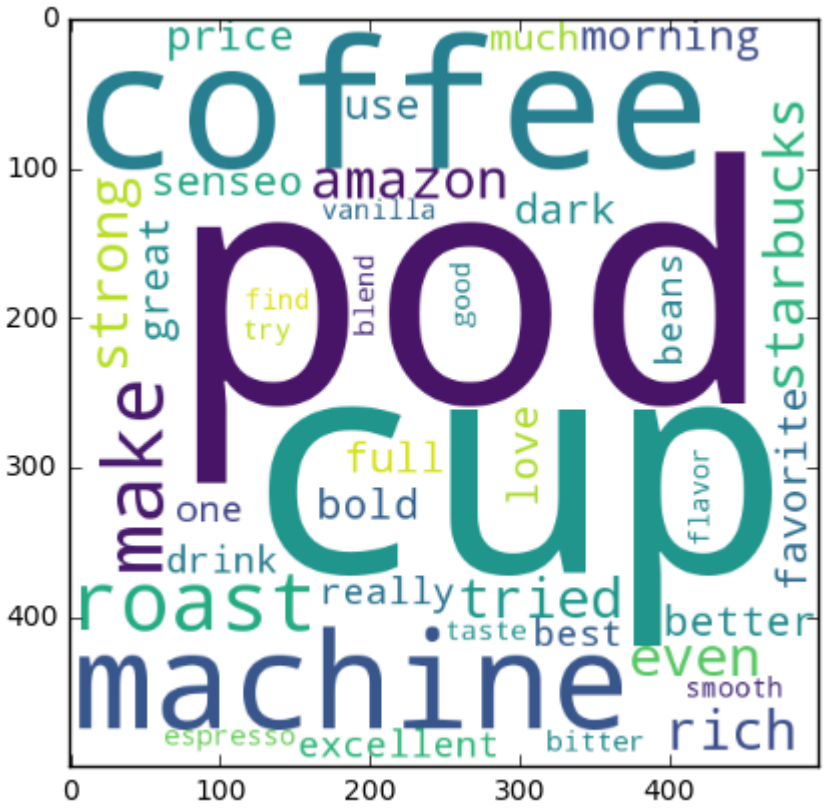
In [338]:
```
plt.imshow(wordcloud_cluster5)
plt.tight_layout(pad = 0)
plt.show()
```



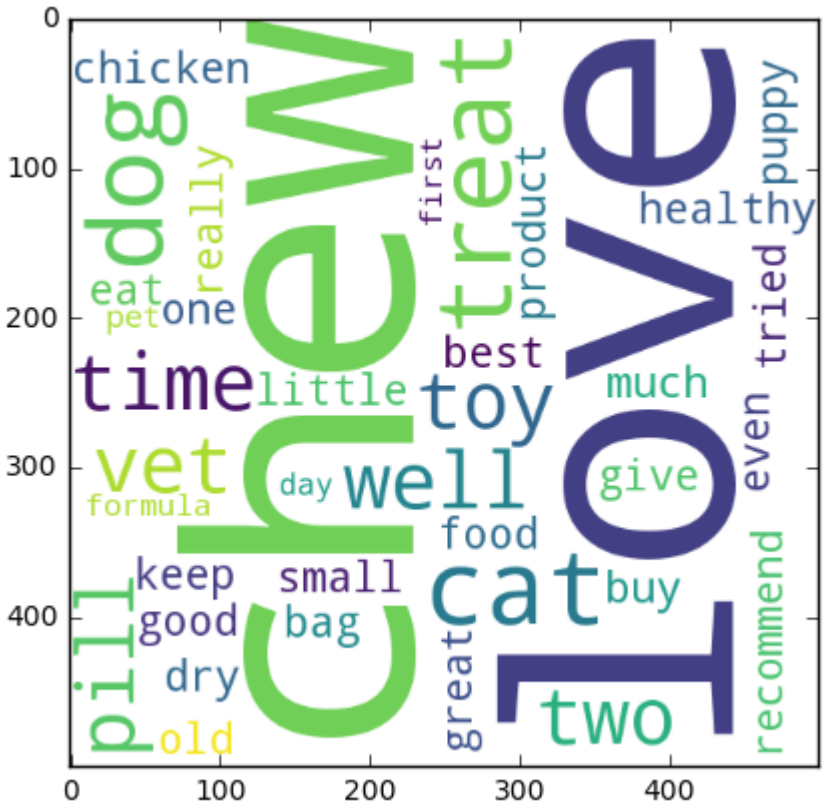
In [339]:
```
plt.imshow(wordcloud_cluster6)
plt.tight_layout(pad = 0)
plt.show()
```



In [340]:
```
plt.imshow(wordcloud_cluster7)
plt.tight_layout(pad = 0)
plt.show()
```



In [341]:
```
plt.imshow(wordcloud_cluster8)
plt.tight_layout(pad = 0)
plt.show()
```



Agglomorative Clustering

In [14]:
```
X_train = X[0:5000]
Y_train = y[0:5000]
X_val = X[5000:7000]
Y_val = y[5000:7000]
X_test = X[7000:9000]
Y_test = y[7000:9000]
```

In [15]:
```
print(len(X_train), len(X_test), len(X_val))
print(len(Y_train), len(Y_test), len(Y_val))
```

```
5000 2000 2000
5000 2000 2000
```

In [117]:
```
avg_w2v_train_5k = avg_w2vec([sent.split() for sent in X_train])
```

```
100%|████████████████████████████████████████████████████████████
████████████████████████████| 5000/5000 [01:19<00:00, 62.72it/s]

5000
50
```

In [287]:
```
with open('avg_w2v_train_5k.pkl', 'wb') as handle:
    pickle.dump(avg_w2v_train_5k, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [27]:
```
tfidf_w2v_train = tfidf_w2v([sent.split() for sent in X_train])
```

```
100%|████████████████████████████████████████████████████████████
████████████████████████| 5000/5000 [05:24<00:00, 11.98it/s]
```

In [28]:
```
with open('tfidf_w2v_train_5k.pkl', 'wb') as handle:
    pickle.dump(tfidf_w2v_train, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Agglomorative clustering on Avg-w2v

In [293]:
```
from sklearn.cluster import AgglomerativeClustering
from tqdm import tqdm

avg_agg_k_inertia = {}

avg_agg_clf = AgglomerativeClustering(n_clusters = 2)
avg_agg_clf.fit(avg_w2v_train_5k)
```

Out[293]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=2,
            pooling_func='deprecated')
```

In [294]:
```
avg_agg_clf.get_params
```

Out[294]:
```
<bound method BaseEstimator.get_params of AgglomerativeClustering(affinity='euc
lidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=2,
            pooling_func='deprecated')>
```

In [295]:
```
#Labels of each point
avg_agg_clf.labels_
mydict = {i: np.where(avg_agg_clf.labels_ == i)[0] for i in range(avg_agg_clf.n_c
# Transform this dictionary into list (if you need a list as result)
dict_list = []
key_list = []
value_list = []
for key, value in mydict.items():
    key_list.append(key)
    value_list.append(value)
complete_dict = dict(zip(key_list, value_list))
```

In [296]:
```
complete_dict
```

Out[296]:
```
{0: array([   0,    1,    2, ..., 4996, 4998, 4999], dtype=int64),
 1: array([  28,   40,   48, ..., 4978, 4984, 4997], dtype=int64)}
```

In [297]:
```
for index, values in complete_dict.items():
    print(index, len(values), values)
```

```
0 3955 [   0    1    2 ... 4996 4998 4999]
1 1045 [  28   40   48 ... 4978 4984 4997]
```

In [298]:
```
all_dict = {}
for index, values in complete_dict.items():
    review_list = []
    for individual_review in values:
        review_list.append(X_train.iloc[individual_review])
    all_dict[index] = review_list
```

In [299]:
```
# [x for xs in lst for x in xs.split(',')]
words_dict = {}
for key in all_dict.keys():
    r = " ".join(all_dict[key])
    words_dict[key] = r
```

In [300]:
```
final_dict = {}
for other_key in words_dict.keys():
    g = words_dict[other_key].split(' ')
    final_dict[other_key] = g
```

In [308]:
```
from collections import Counter
cluster_dict = {}
for k in final_dict.keys():
    top_words = []
    c = Counter(final_dict[k])
    for i,j in c.most_common(100):
        top_words.append(i)
    cluster_dict[k] = top_words
```

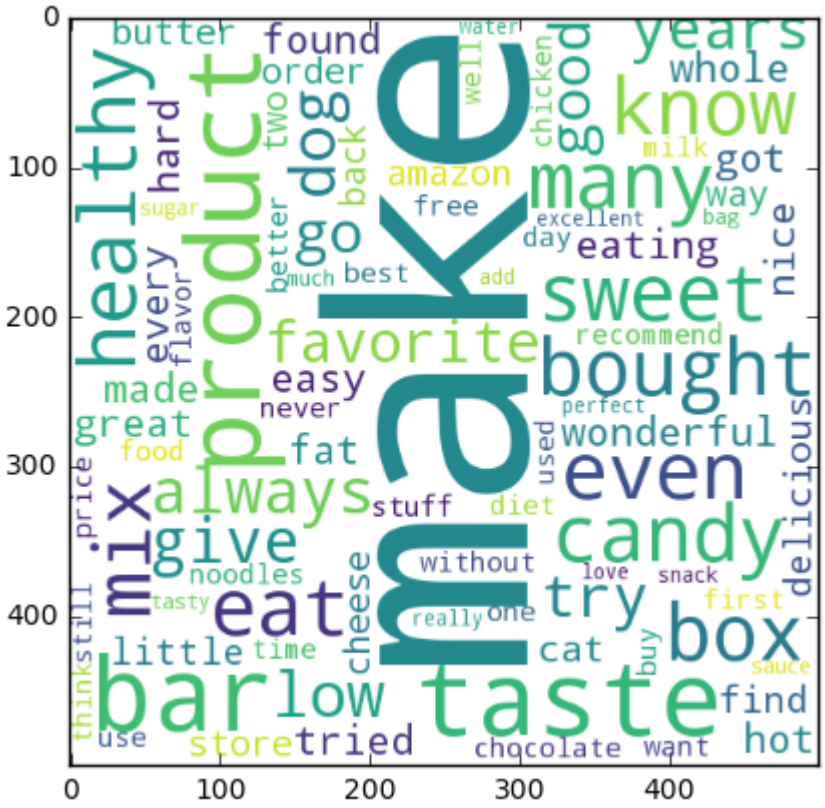
In [309]:
```
cluster_dict[0]
```

Out[309]:
```
['not',
 'like',
 'good',
 'great',
 'one',
 'taste',
 'product',
 'best',
 'would',
 'love',
 'flavor',
 'chocolate',
 'no',
 'get',
 'really',
 'food',
 'also',
 'eat',
 'little',
 'even',
 'make',
 'time',
 'much',
 'use',
 'find',
 'well',
 'better',
 'try',
 'sauce',
 'cheese',
 'buy',
 'tried',
 'hot',
 'delicious',
 'sugar',
 'ever',
 'used',
 'amazon',
 'made',
 'first',
 'sweet',
 'found',
 'stuff',
 'candy',
 'could',
 'cat',
 'mix',
 'since',
 'think',
 'go',
 'way',
 'box',
 'price',
 'never',
 'years',
```

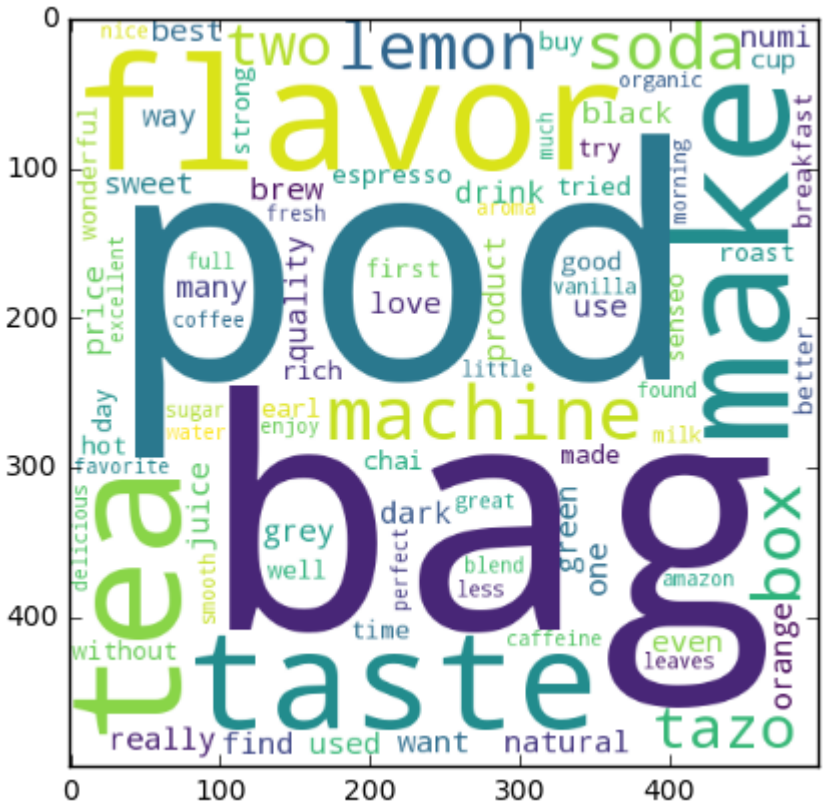
```
'many',
'excellent',
'favorite',
'snack',
'milk',
'day',
'two',
'order',
'free',
'still',
'store',
'know',
'healthy',
'makes',
'tasty',
'fat',
'bars',
'wonderful',
'add',
'easy',
'want',
'eating',
'water',
'whole',
'bought',
'recommend',
'dog',
'products',
'give',
'without',
'bar',
'butter',
'tastes',
'chicken',
'nice',
'always',
'low',
'noodles',
'perfect',
'bag',
'diet',
'back',
'got',
'hard',
'every']
```

In [312]:
```
cl1_string = ' '.join(cluster_dict[0])
cl2_string = ' '.join(cluster_dict[1])
```

In [313]:
```
from wordcloud import WordCloud

wordcloud_cluster1 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster2 = WordCloud(width = 500, height = 500, background_color ='whit

# plot the WordCloud image
plt.imshow(wordcloud_cluster1)
plt.tight_layout(pad = 0)
plt.show()
```



In [314]:
```
plt.imshow(wordcloud_cluster2)
plt.tight_layout(pad = 0)
plt.show()
```



Agglomoretive clustering with k = 5

In [348]:
```
from sklearn.cluster import AgglomerativeClustering
from tqdm import tqdm

avg_agg_k_inertia = {}

avg_agg_clf = AgglomerativeClustering(n_clusters = 5)
avg_agg_clf.fit(avg_w2v_train_5k)
```

Out[348]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=5,
            pooling_func='deprecated')
```

In [349]:
```
avg_agg_clf.get_params
```

Out[349]:
```
<bound method BaseEstimator.get_params of AgglomerativeClustering(affinity='euc
lidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=5,
            pooling_func='deprecated')>
```

In [350]:
```
#Labels of each point
avg_agg_clf.labels_
mydict = {i: np.where(avg_agg_clf.labels_ == i)[0] for i in range(avg_agg_clf.n_c
# Transform this dictionary into list (if you need a list as result)
dict_list = []
key_list = []
value_list = []
for key, value in mydict.items():
    key_list.append(key)
    value_list.append(value)
complete_dict = dict(zip(key_list, value_list))
```

In [353]:
```
all_dict = {}
for index, values in complete_dict.items():
    review_list = []
    for individual_review in values:
        review_list.append(X_train.iloc[individual_review])
    all_dict[index] = review_list
```

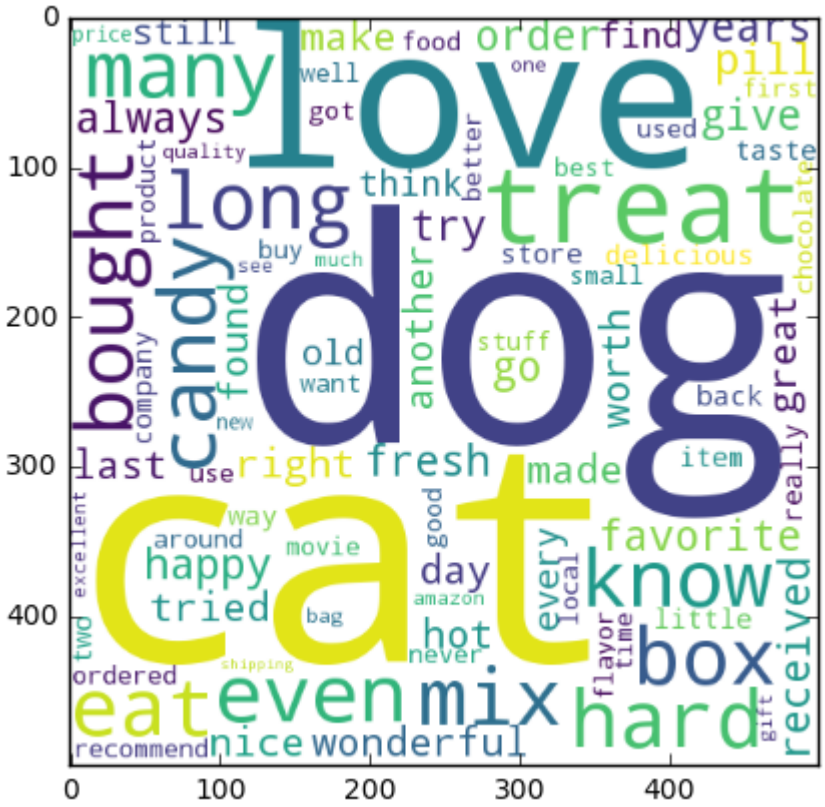
In [354]:
```
# [x for xs in lst for x in xs.split(',')]
words_dict = {}
for key in all_dict.keys():
    r = " ".join(all_dict[key])
    words_dict[key] = r
```

In [355]:
```
final_dict = {}
for other_key in words_dict.keys():
    g = words_dict[other_key].split(' ')
    final_dict[other_key] = g
```

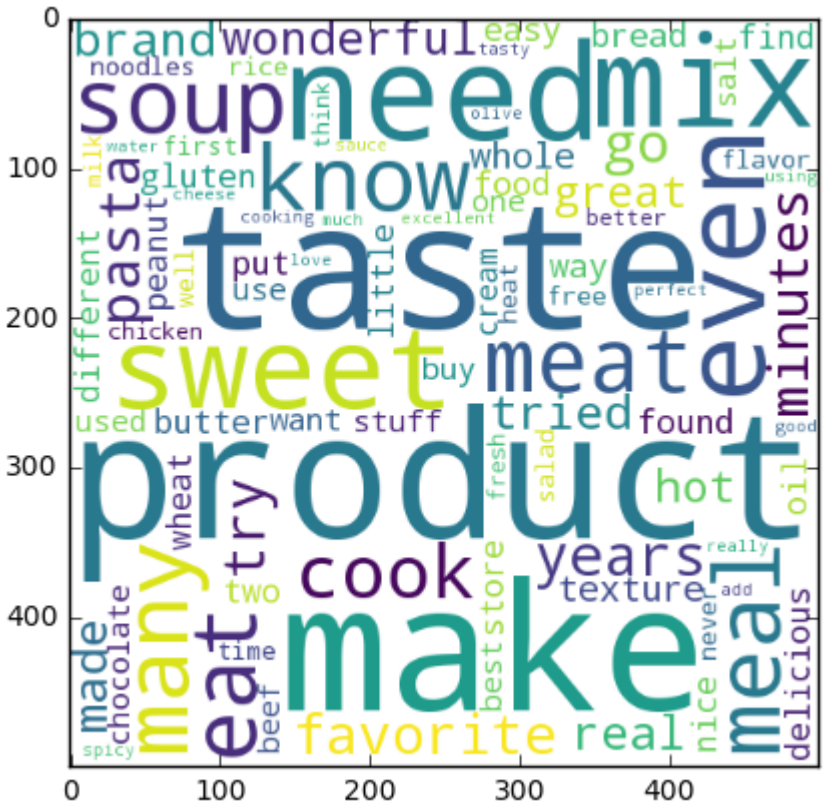
In [356]:
```
from collections import Counter
cluster_dict = {}
for k in final_dict.keys():
    top_words = []
    c = Counter(final_dict[k])
    for i,j in c.most_common(100):
        top_words.append(i)
    cluster_dict[k] = top_words
```

In [357]:
```
cl1_string = ' '.join(cluster_dict[0])
cl2_string = ' '.join(cluster_dict[1])
cl3_string = ' '.join(cluster_dict[2])
cl4_string = ' '.join(cluster_dict[3])
cl5_string = ' '.join(cluster_dict[4])
```

In [358]:
```
from wordcloud import WordCloud

wordcloud_cluster1 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster2 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster3 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster4 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster5 = WordCloud(width = 500, height = 500, background_color ='whit

# plot the WordCloud image
plt.imshow(wordcloud_cluster1)
plt.tight_layout(pad = 0)
plt.show()
```

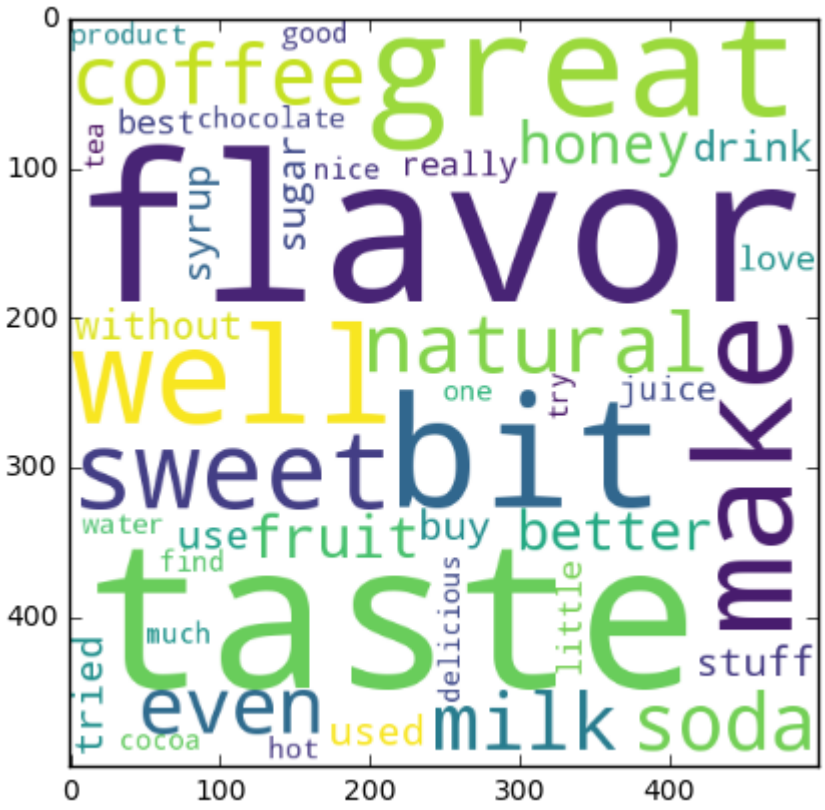


In [359]:
```
plt.imshow(wordcloud_cluster2)
plt.tight_layout(pad = 0)
plt.show()
```



In [360]:
```
plt.imshow(wordcloud_cluster3)
plt.tight_layout(pad = 0)
plt.show()
```

In [361]:
```
plt.imshow(wordcloud_cluster4)
plt.tight_layout(pad = 0)
plt.show()
```



In [362]:
```
plt.imshow(wordcloud_cluster5)
plt.tight_layout(pad = 0)
plt.show()
```

Agglomerative clustering on tfidf-w2v

In [365]:
```
from sklearn.cluster import AgglomerativeClustering
from tqdm import tqdm

tfw2v_agg_k_inertia = {}

tfw2v_agg_clf = AgglomerativeClustering(n_clusters = 2)
tfw2v_agg_clf.fit(tfidf_w2v_train)
```

Out[365]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=2,
            pooling_func='deprecated')
```

In [366]:
```
tfw2v_agg_clf.get_params
```

Out[366]:
```
<bound method BaseEstimator.get_params of AgglomerativeClustering(affinity='euc
lidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=2,
            pooling_func='deprecated')>
```

In [367]:
```
#Labels of each point
tfw2v_agg_clf.labels_
mydict = {i: np.where(tfw2v_agg_clf.labels_ == i)[0] for i in range(tfw2v_agg_clf
# Transform this dictionary into list (if you need a list as result)
dict_list = []
key_list = []
value_list = []
for key, value in mydict.items():
    key_list.append(key)
    value_list.append(value)
complete_dict = dict(zip(key_list, value_list))
```

In [370]:
```
all_dict = {}
for index, values in complete_dict.items():
    review_list = []
    for individual_review in values:
        review_list.append(X_train.iloc[individual_review])
    all_dict[index] = review_list
```

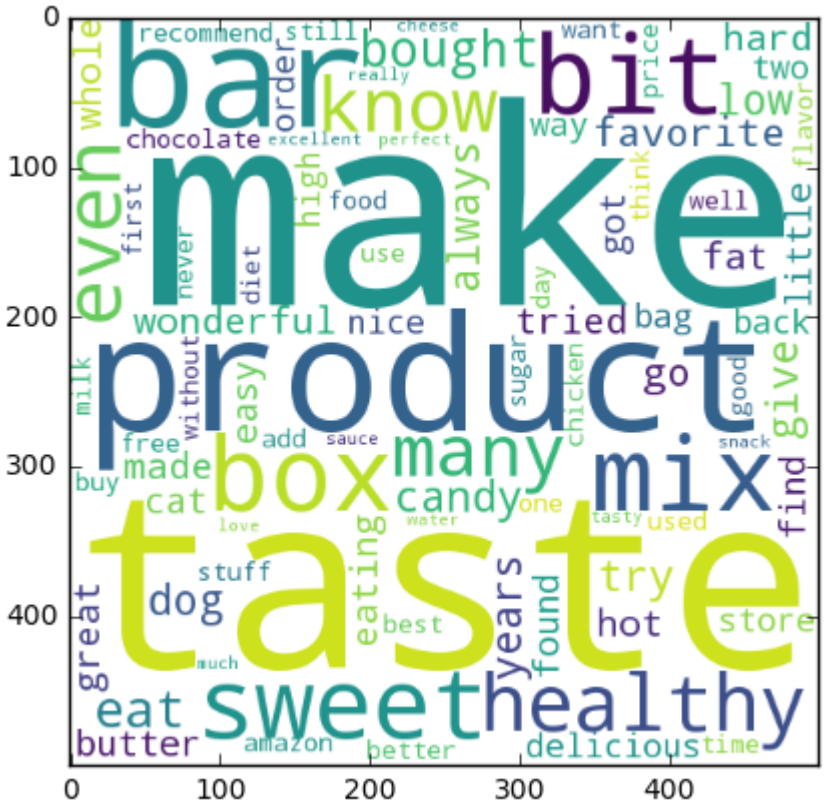
In [371]:
```
# [x for xs in lst for x in xs.split(',')]
words_dict = {}
for key in all_dict.keys():
    r = " ".join(all_dict[key])
    words_dict[key] = r
```

In [372]:
```
final_dict = {}
for other_key in words_dict.keys():
    g = words_dict[other_key].split(' ')
    final_dict[other_key] = g
```

In [373]:
```
from collections import Counter
cluster_dict = {}
for k in final_dict.keys():
    top_words = []
    c = Counter(final_dict[k])
    for i,j in c.most_common(100):
        top_words.append(i)
    cluster_dict[k] = top_words
```

In [375]:
```
cl1_string = ' '.join(cluster_dict[0])
cl2_string = ' '.join(cluster_dict[1])
```

In [376]:
```
from wordcloud import WordCloud

wordcloud_cluster1 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster2 = WordCloud(width = 500, height = 500, background_color ='whit

# plot the WordCloud image
plt.imshow(wordcloud_cluster1)
plt.tight_layout(pad = 0)
plt.show()
```



In [377]:
```
plt.imshow(wordcloud_cluster2)
plt.tight_layout(pad = 0)
plt.show()
```

Agglomoretive clustering with k = 5

In [378]:
```
from sklearn.cluster import AgglomerativeClustering
from tqdm import tqdm

tfw2v_agg_clf = AgglomerativeClustering(n_clusters = 5)
tfw2v_agg_clf.fit(tfidf_w2v_train)
```

Out[378]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=5,
            pooling_func='deprecated')
```

In [379]:
```
avg_agg_clf.get_params
```

Out[379]:
```
<bound method BaseEstimator.get_params of AgglomerativeClustering(affinity='euc
lidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=5,
            pooling_func='deprecated')>
```

In [380]:
```
#Labels of each point
avg_agg_clf.labels_
mydict = {i: np.where(avg_agg_clf.labels_ == i)[0] for i in range(avg_agg_clf.n_c
# Transform this dictionary into list (if you need a list as result)
dict_list = []
key_list = []
value_list = []
for key, value in mydict.items():
    key_list.append(key)
    value_list.append(value)
complete_dict = dict(zip(key_list, value_list))
```

In [382]:
```
all_dict = {}
for index, values in complete_dict.items():
    review_list = []
    for individual_review in values:
        review_list.append(X_train.iloc[individual_review])
    all_dict[index] = review_list
```

In [383]:
```
# [x for xs in lst for x in xs.split(',')]
words_dict = {}
for key in all_dict.keys():
    r = " ".join(all_dict[key])
    words_dict[key] = r
```

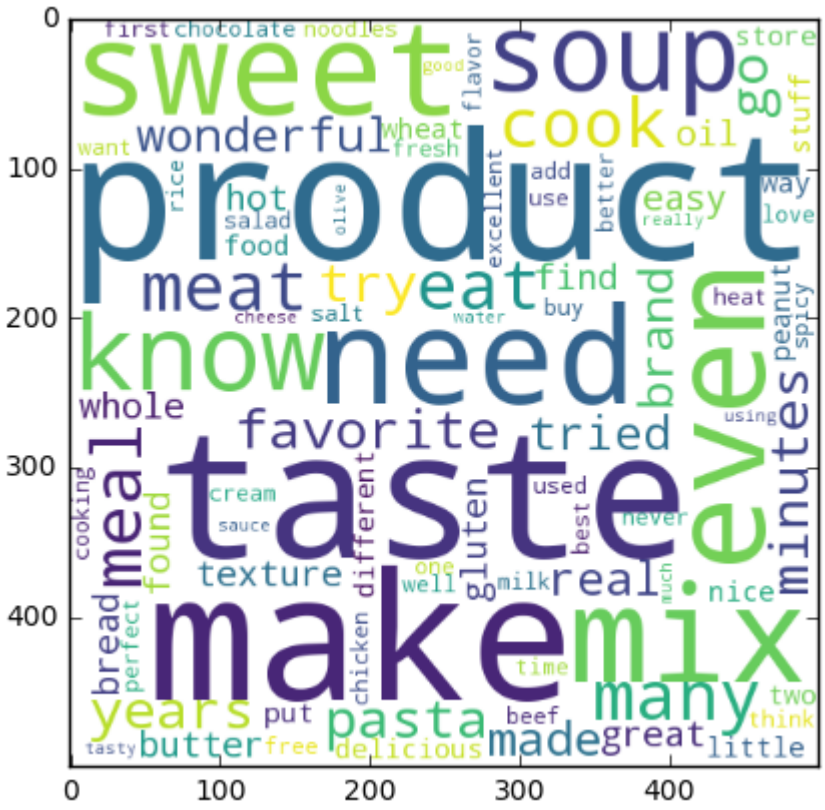
In [384]:
```
final_dict = {}
for other_key in words_dict.keys():
    g = words_dict[other_key].split(' ')
    final_dict[other_key] = g
```

In [385]:
```
from collections import Counter
cluster_dict = {}
for k in final_dict.keys():
    top_words = []
    c = Counter(final_dict[k])
    for i,j in c.most_common(100):
        top_words.append(i)
    cluster_dict[k] = top_words
```

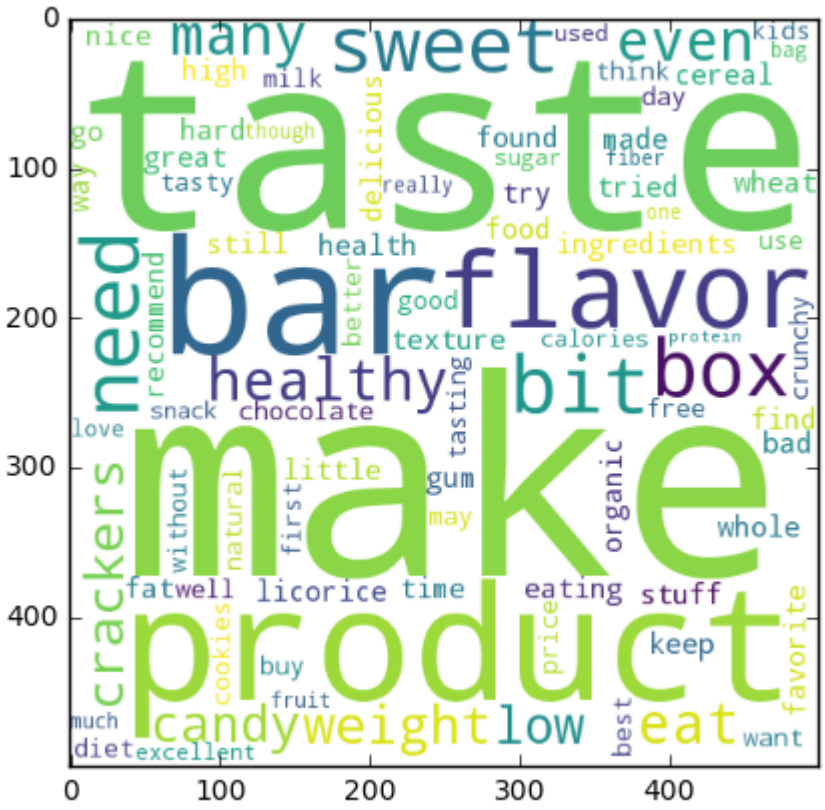
In [386]:
```
cl1_string = ' '.join(cluster_dict[0])
cl2_string = ' '.join(cluster_dict[1])
cl3_string = ' '.join(cluster_dict[2])
cl4_string = ' '.join(cluster_dict[3])
cl5_string = ' '.join(cluster_dict[4])
```

In [387]:
```
from wordcloud import WordCloud

wordcloud_cluster1 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster2 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster3 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster4 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster5 = WordCloud(width = 500, height = 500, background_color ='whit

# plot the WordCloud image
plt.imshow(wordcloud_cluster1)
plt.tight_layout(pad = 0)
plt.show()
```

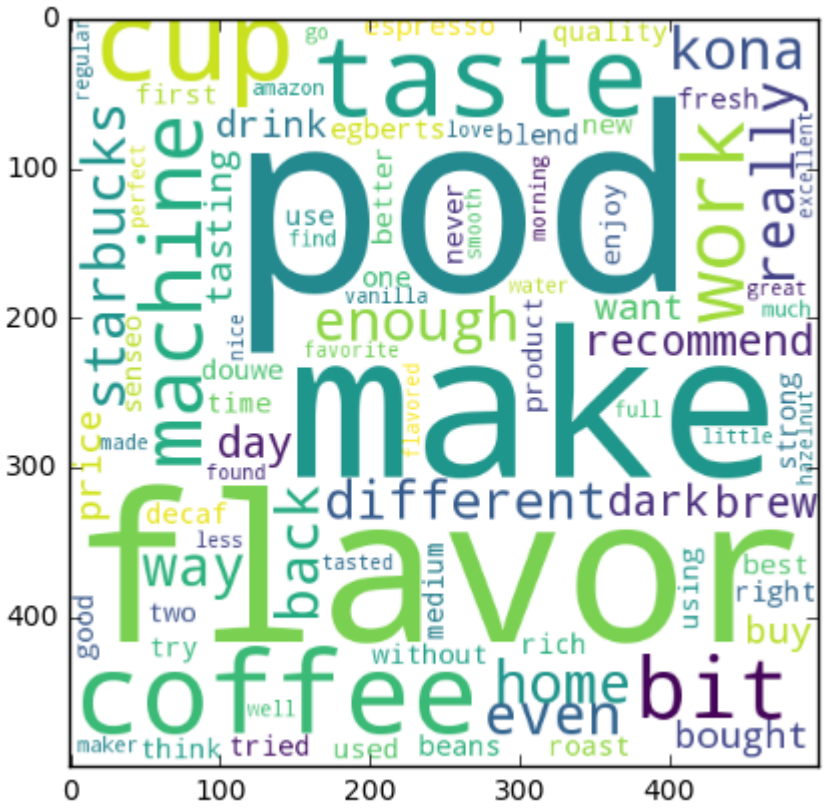
In [388]:
```
plt.imshow(wordcloud_cluster2)
plt.tight_layout(pad = 0)
plt.show()
```



In [389]:
```
plt.imshow(wordcloud_cluster3)
plt.tight_layout(pad = 0)
plt.show()
```



In [390]:
```
plt.imshow(wordcloud_cluster4)
plt.tight_layout(pad = 0)
plt.show()
```



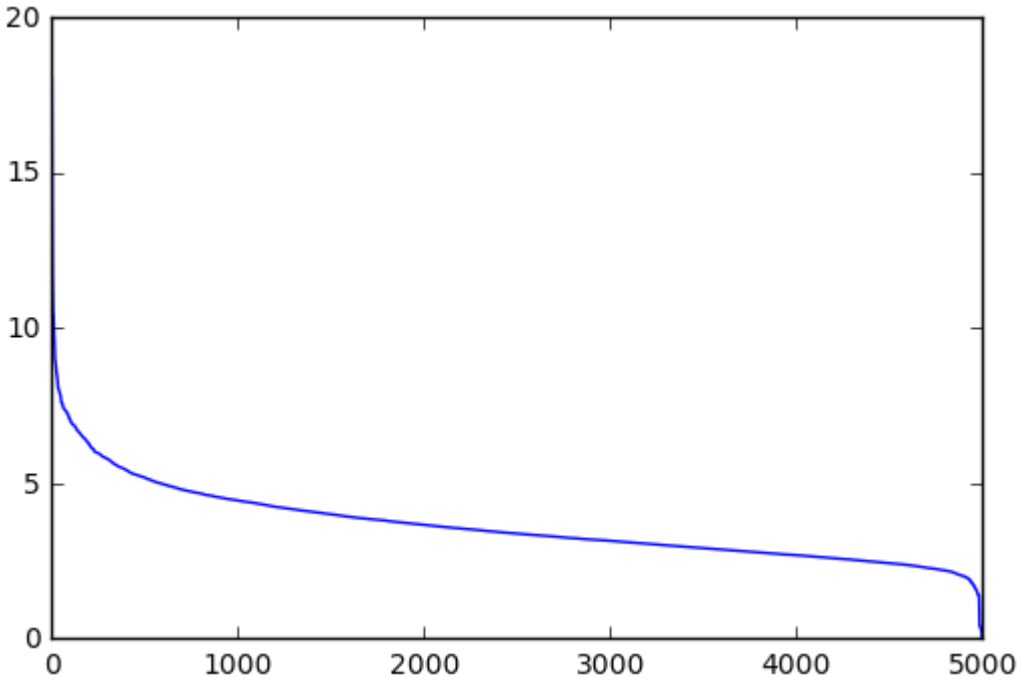
In [391]:
```
plt.imshow(wordcloud_cluster5)
plt.tight_layout(pad = 0)
plt.show()
```

DBSCAN on Avg-w2v

In [166]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler(with_mean=False)
std_avg_w2v_train_5k=scaler.fit_transform(avg_w2v_train_5k)
```

In [167]:
```
from sklearn.neighbors import NearestNeighbors
#https://stackoverflow.com/questions/48010276/how-to-estimate-eps-using-knn-distar

ns = 4
nbrs = NearestNeighbors(n_neighbors=ns).fit(std_avg_w2v_train_5k)
distances, indices = nbrs.kneighbors(std_avg_w2v_train_5k)
distanceDec = sorted(distances[:,ns-1], reverse=True)
plt.plot(list(range(1,len(avg_w2v_train_5k)+1)), distanceDec)
```

Out[167]:
```
[<matplotlib.lines.Line2D at 0x239cacc1160>]
```



Epsilon = 3

In [168]:
```
from sklearn.cluster import DBSCAN

avg_clustering = DBSCAN(eps=3, min_samples=10)
avg_clustering.fit(std_avg_w2v_train_5k)
```

Out[168]:
```
DBSCAN(algorithm='auto', eps=3, leaf_size=30, metric='euclidean',
    metric_params=None, min_samples=10, n_jobs=None, p=None)
```

In [169]:
```
cluster_dict = {}
print(len(avg_clustering.labels_))
print(avg_clustering.labels_)
for ind_label in avg_clustering.labels_:
    cluster_dict[ind_label] = X_train
```

```
5000
[ 1  1 -1 ... -1 -1 -1]
```

In [170]:
```
core_samples = avg_clustering.core_sample_indices_
labels = avg_clustering.labels_
```

In [171]:
```
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print(n_clusters_)
unique_labels = set(labels)
print(unique_labels)
```

```
2
{0, 1, -1}
```

In [172]:
```
all_clusters = {}
for k in unique_labels:
#     index_list = []
    class_members = [index[0] for index in np.argwhere(labels == k)]
    #cluster_core_samples = [index for index in core_samples if labels[index] == k
#     print(class_members)
    all_clusters[k] = class_members

#     for index in class_members:
#         x = std_tfidf_w2v_train_5k[index]
#     print(x)
```

In [173]:
```
all_dict = {}
for index, values in all_clusters.items():
    review_list = []
    for individual_review in values:
        review_list.append(X_train.iloc[individual_review])
    all_dict[index] = review_list
```

In [174]:
```
words_dict = {}
for key in all_dict.keys():
    r = " ".join(all_dict[key])
    words_dict[key] = r
```

In [175]:
```
final_dict = {}
for other_key in words_dict.keys():
    g = words_dict[other_key].split(' ')
    final_dict[other_key] = g
```

In [179]:
```
from collections import Counter
avg_cluster_dict = {}
for k in final_dict.keys():
    top_words = []
    c = Counter(final_dict[k])
    for i,j in c.most_common(100):
        top_words.append(i)
    avg_cluster_dict[k] = top_words
```

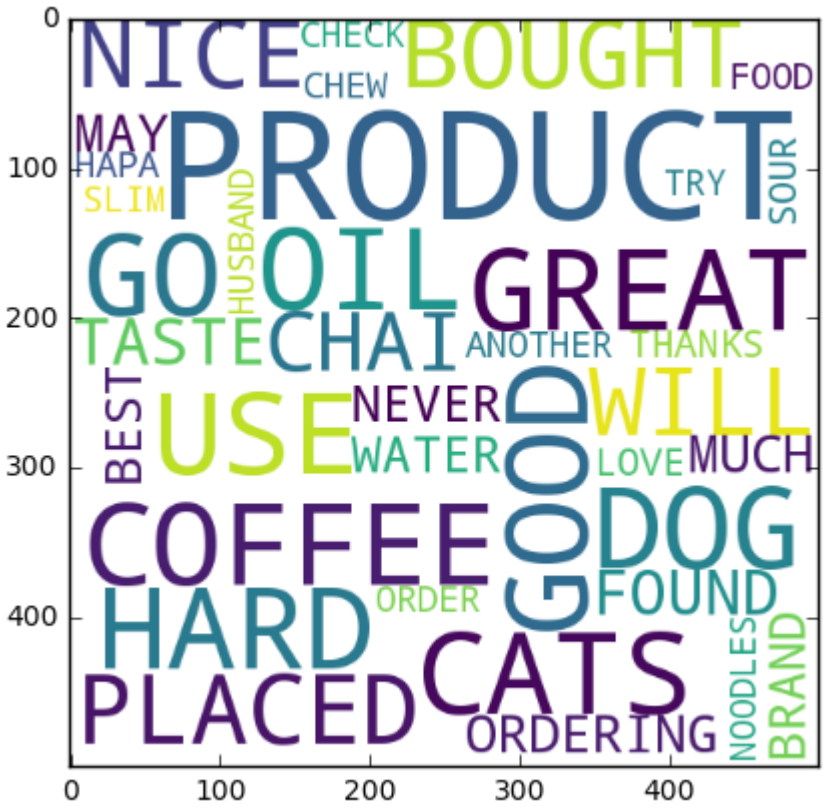
In [180]:
```
avg_cluster_dict.keys()
```

Out[180]:
```
dict_keys([0, 1, -1])
```

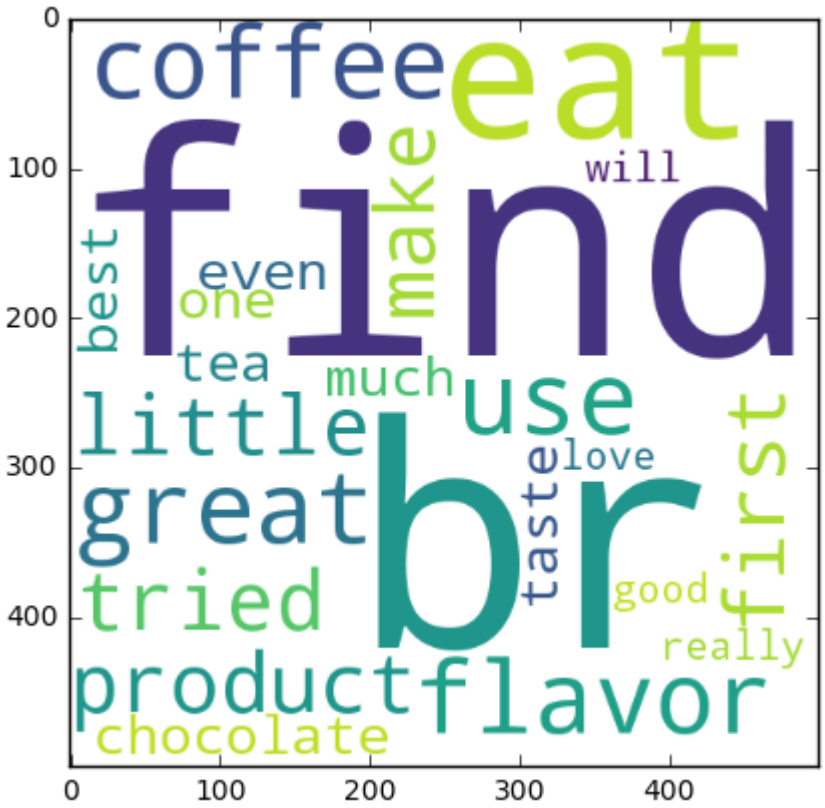
In [182]:
```
cl1_string = ' '.join(avg_cluster_dict[0])
cl2_string = ' '.join(avg_cluster_dict[1])
```

In [183]:
```
from wordcloud import WordCloud

wordcloud_cluster1 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster2 = WordCloud(width = 500, height = 500, background_color ='whit
```

In [184]:
```
# plot the WordCloud image
plt.imshow(wordcloud_cluster1)
plt.tight_layout(pad = 0)
plt.show()
```



In [185]:
```
# plot the WordCloud image
plt.imshow(wordcloud_cluster2)
plt.tight_layout(pad = 0)
plt.show()
```

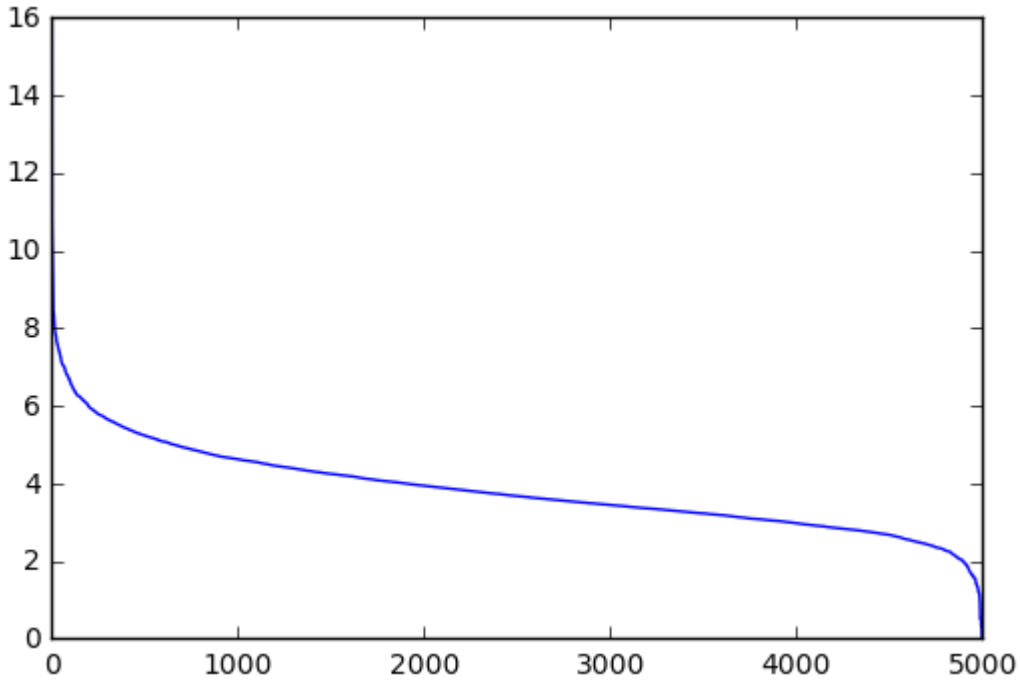


DBSCAN on Tfidf-w2v

In [30]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler(with_mean=False)
std_tfidf_w2v_train_5k=scaler.fit_transform(tfidf_w2v_train)
```

In [500]:
```
from sklearn.neighbors import NearestNeighbors
#https://stackoverflow.com/questions/48010276/how-to-estimate-eps-using-knn-distar

ns = 4
nbrs = NearestNeighbors(n_neighbors=ns).fit(std_tfidf_w2v_train_5k)
distances, indices = nbrs.kneighbors(std_tfidf_w2v_train_5k)
distanceDec = sorted(distances[:,ns-1], reverse=True)
plt.plot(list(range(1,len(std_tfidf_w2v_train_5k)+1)), distanceDec)
```

Out[500]:
```
[<matplotlib.lines.Line2D at 0x72294940>]
```



Epsilon = 3

In [62]:
```
from sklearn.cluster import DBSCAN

tfidfw2v_clustering = DBSCAN(eps=3, min_samples=10)
tfidfw2v_clustering.fit(std_tfidf_w2v_train_5k)
```

Out[62]:
```
DBSCAN(algorithm='auto', eps=3, leaf_size=30, metric='euclidean',
    metric_params=None, min_samples=10, n_jobs=None, p=None)
```

In [78]:
```
core_samples = tfidfw2v_clustering.core_sample_indices_
labels = tfidfw2v_clustering.labels_
```

In [82]:
```
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print(n_clusters_)
unique_labels = set(labels)
print(unique_labels)
```

```
2
{0, 1, -1}
```

In [100]:
```
all_clusters = {}
for k in unique_labels:
#     index_list = []
    class_members = [index[0] for index in np.argwhere(labels == k)]
    #cluster_core_samples = [index for index in core_samples if labels[index] == k
#     print(class_members)
    all_clusters[k] = class_members

#     for index in class_members:
#         x = std_tfidf_w2v_train_5k[index]
#     print(x)
```

In [104]:
```
all_dict = {}
for index, values in all_clusters.items():
    review_list = []
    for individual_review in values:
        review_list.append(X_train.iloc[individual_review])
    all_dict[index] = review_list
```

In [64]:
```
cluster_dict = {}
print(len(tfidfw2v_clustering.labels_))
print(tfidfw2v_clustering.labels_)
for ind_label in tfidfw2v_clustering.labels_:
    cluster_dict[ind_label] = X_train
```

```
5000
[ 1 -1 -1 ... -1 -1 -1]
```

In [105]:
```
words_dict = {}
for key in all_dict.keys():
    r = " ".join(all_dict[key])
    words_dict[key] = r
```

In [106]:
```
final_dict = {}
for other_key in words_dict.keys():
    g = words_dict[other_key].split(' ')
    final_dict[other_key] = g
```

In [107]:
```
from collections import Counter
cluster_dict = {}
for k in final_dict.keys():
    top_words = []
    c = Counter(final_dict[k])
    for i,j in c.most_common(100):
        top_words.append(i)
    cluster_dict[k] = top_words
```

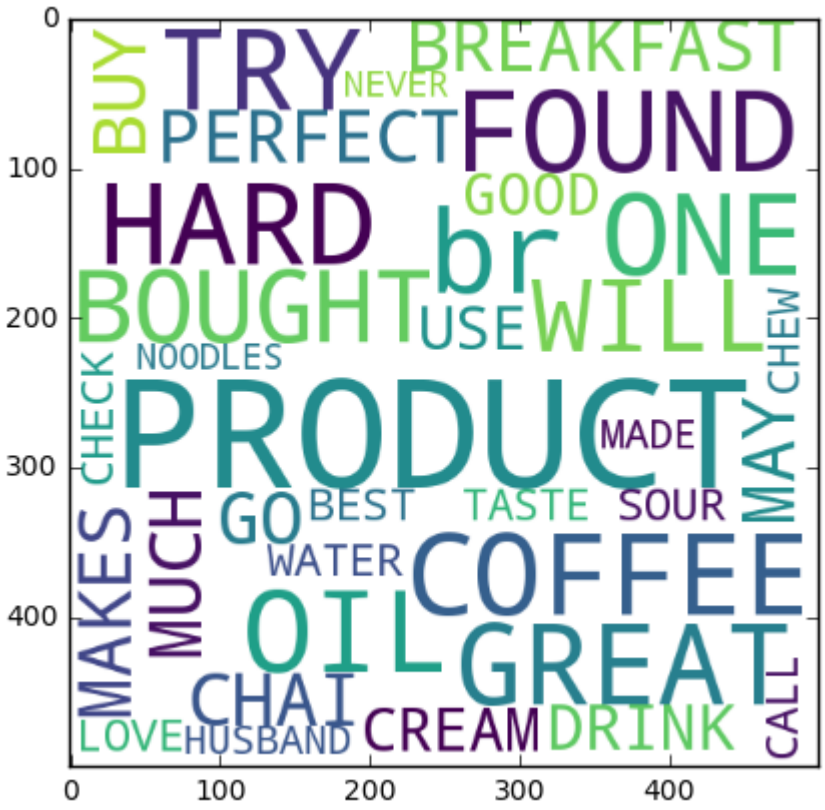
In [108]:
```
cluster_dict.keys()
```

Out[108]:
```
dict_keys([0, 1, -1])
```

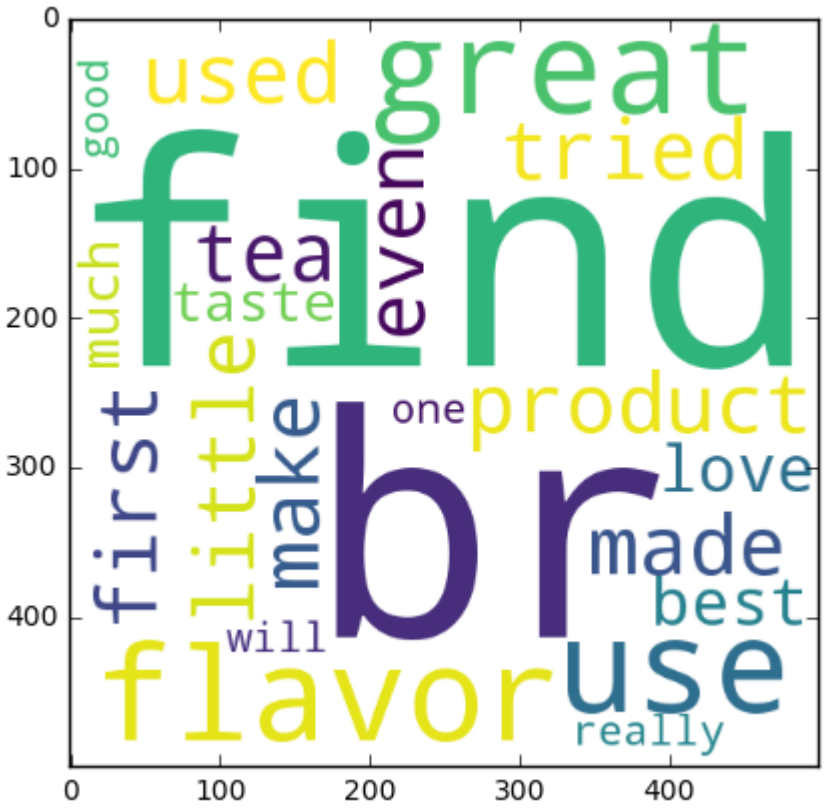
In [109]:
```
cl1_string = ' '.join(cluster_dict[0])
cl2_string = ' '.join(cluster_dict[1])
```

In [111]:
```
from wordcloud import WordCloud

wordcloud_cluster1 = WordCloud(width = 500, height = 500, background_color ='whit
wordcloud_cluster2 = WordCloud(width = 500, height = 500, background_color ='whit
```

In [112]:
```
# plot the WordCloud image
plt.imshow(wordcloud_cluster1)
plt.tight_layout(pad = 0)
plt.show()
```



In [113]:
```
plt.imshow(wordcloud_cluster2)
plt.tight_layout(pad = 0)
plt.show()
```



Observations from K-means BoW clusters

cluster 1 - represents taste and flavours eg.great, taste
cluster 2 - represents products eg. food, flavour
cluster 3 - represents taste of coffee eg. coffee, cup, good
cluster 4 - reprents product adjectives eg.love, best
cluster 5 - represents taste of products eg. product, best
cluster 6 - represents qualities of tea eg. ea, green, organic

Observations from K-means tfidf clusters

cluster 1 - represents fellings of user eg. thirsty, odors
cluster 2 - represents desire eg. impulse, irresistible
cluster 3 - represents type of coffee and its taste eg. arabica, yummy
cluster 4 - reprents problems with food eg. thawed, issues
cluster 5 - represents place eg. Yorkshire
cluster 6 - represents everything about chocolates eg.shavings, hazzlenut

Observations from K-means Avg-w2v clusters

cluster 1 - represents users order detail eg. shipping, Amazon, order, price
cluster 2 - represents qualities of sweet food eg. sugar, gluten, delicioius, bars, healthy
cluster 3 - represents users buying experience eg.buy, well, love, time, find
cluster 4 - reprents food names eg.chicken, oil, sauce, rice
cluster 5 - represents chocolate qualities eg. chocolate, sugar, great, sweet
cluster 6 - represents tea and its qualities eg. tea, green, grey, organic
cluster 7 - represents coffee and its qualities eg. coffee, roast, espresso, strong
cluster 8 - represents animal food eg. dog, cat, eat

Observations from K-means tfidf-w2v clusters

cluster 1 - represents kind of delivery eg. gift, bag, item, received
cluster 2 - represents food names eg. cheese, chicken, sauce, oil
cluster 3 - represents beverages eg.chai, coffee, tea, drink
cluster 4 - represents liquid products eg. milk, water, coffee, juice
cluster 5 - represents breakfast items eg. cereals, chips, snack, cookies, bar
cluster 6 - represents experience of first time buyers eg. time, first, well, amazon, years
cluster 7 - represents everything about coffee eg, coffee, starbucks, strong, espresso, rich
cluster 8 - represents animal food eg, dog, chew, pupy, healthy, vet

Observations from Agglomerative Avg-w2v clusters with n =5

cluster 1 - represents animal food and users experience eg. dog, cat, happy, nice, recommend
cluster 2 - represents food names and its cooking ways eg. pasta, butter, bread, peanut, meat
cluster 3 - represents liquid items eg. water, chai, tea, drink
cluster 4 - represents coffe items eg. coffee, milk, cocoa,
cluster 5 - represents breakfast items eg. cereals, chips, snack, cookies, bar

Observations from Agglomerative tfidf-w2v clusters with n =5

cluster 1 - represents reactions eg. hot, fresh, good
cluster 2 - represents product details eg. product, tasta, make, mix
cluster 3 - represents product health details eg, healthy, calories, protein
cluster 4 - represents coffee details eg. beans, starbucks, machine, cup
cluster 5 - represents liquid items eg. tea, soda, coffee

Observations from DBSCAN with Avg_w2v

cluster 1 - represents food names eg. chai, noodles, coffee
cluster 2 - represents taste of food eg. great, tried, best

Observations from DBSCAN with tfidf_w2v

cluster 1 - represents food qualities eg. best, perfect, good
cluster 2 - represents usage of user eg. little, first, tried

Steps followed to complete this assignment

K-Means Clustering

1. Performed data cleaning and created Bag of Words(BoW), Tf-idf, Avg-w2v, tfidf-w2v
2. Considered n_clusters in the range of 1 to 15 to find the best value of n_clusters
3. Plotted inertia vs n_clusters graph to determine the best value of n using elbow-knee method
4. Once best n is obtained trained the K-means on that value.
5. Created wordclouds using the feature names of vectorizer and centroids

Agglomerative Clustering

1. Performed data cleaning on 5000 data points and created Bag of Words(BoW), Tf-idf, Avg-w2v, tfidf-w2v
2. Considered n = 2 and n = 5 as random number of clusters to train the model
3. Got the index of reviews belonging to particular review and extracted words from it make wordcloud
4. Created wordcloud of the feature names extracted of n number of clusters

DBSCAN clustering

1. Performed data cleaning on 5000 data points and created Bag of Words(BoW), Tf-idf, Avg-w2v, tfidf-w2v
2. To find the value of epsilon used KNN distance and plotted the graph
3. After obtaining the best epsilon trained the model with epsilon value with min_samples = 10
4. Obtained the indices of reviews rows of particular cluster and extraceted the unique features from those
5. Plotted the wordcloud of features obtained