

# Amazon Fine Food Reviews Analysis

Data Source: <https://www.kaggle.com/snap/amazon-fine-food-reviews>

EDA: <https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/>

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454

Number of users: 256,059

Number of products: 74,258

Timespan: Oct 1999 - Oct 2012

Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unique identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

## Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use Score/Rating. A rating of 4 or 5 can be considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

## [1]. Reading Data

### [1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation will be set to "positive". Otherwise, it will be set to "negative".

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 [2]:

```

# using SQLite Table to read data.
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 points
# 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 LIMIT 500000""", co
n)
# for tsne assignment you can take 5k data points

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

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
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[2]:

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

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

In [3]:

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

In [4]:

```
print(display.shape)
display.head()
```

(80668, 7)

Out[4]:

	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 [5]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

In [6]:

```
display['COUNT(*)'].sum()
```

Out[6]:

393063

## [2] Exploratory Data Analysis

### [2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [7]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
```

```
FROM reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANII WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANII WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANII WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANII WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANII WAFE

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delete the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [8]:

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

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[9]:

(364173, 10)

In [10]:

```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]:

69.25890143662969

**Observation:-** It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calculations

In [11]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside

In [12]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

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

(364171, 10)

Out[13]:

```
1    307061
0     57110
Name: Score, dtype: int64
```

## [3] Preprocessing

### [3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.

3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was observed to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [14]:

```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

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 this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

=====

I was really looking forward to these pods based on the reviews. Starbucks is good, but I prefer bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes. I admit that Amazon agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!! I'm hoping to find local San Diego area shoppe that carries pods so that I can try something different than starbucks.

=====

Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belongs in it is Canola oil. Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured out a way to fix that. I still like it but it could be better.

=====

Can't do sugar. Have tried scores of SF Syrups. NONE of them can touch the excellence of this product.<br /><br />Thick, delicious. Perfect. 3 ingredients: Water, Maltitol, Natural Maple Flavor. PERIOD. No chemicals. No garbage.<br /><br />Have numerous friends & family members hooked on this stuff. My husband & son, who do NOT like "sugar free" prefer this over major label regular syrup.<br /><br />I use this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbelievably delicious...<br /><br />Can you tell I like it? :)

=====

In [15]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_1500 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

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 this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

In [16]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup
```

```

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)

```

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 this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

I was really looking forward to these pods based on the reviews. Starbucks is good, but I prefer bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes. I admit that Amazon agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!! I'm hoping to find local San Diego area shoppe that carries pods so that I can try something different than starbucks.

Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belongs in it is Canola oil. Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured out a way to fix that. I still like it but it could be better.

Can't do sugar. Have tried scores of SF Syrups. NONE of them can touch the excellence of this product. Thick, delicious. Perfect. 3 ingredients: Water, Maltitol, Natural Maple Flavor. PERIOD. No chemicals. No garbage. Have numerous friends & family members hooked on this stuff. My husband & son, who do NOT like "sugar free" prefer this over major label regular syrup. I use this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbelievably delicious...Can you tell I like it? :)

In [17]:

```

# 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 [18]:

```

sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)

```

Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belongs in it is Canola oil. Canola or rapeseed is not something a dog would ever fi

I do not think belongs in it is Canola oil. Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it, it would poison them. Today is Food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70 is it was poisonous until they figured out a way to fix that. I still like it but it could be better.

In [19]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

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 this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

In [20]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

Great ingredients although chicken should have been 1st rather than chicken broth the only thing I do not think belongs in it is Canola oil Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it it would poison them Today is Food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut facts though say otherwise Until the late 70 is it was poisonous until they figured out a way to fix that I still like it but it could be better

In [21]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have been removed in the 1st step

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

In [22]:

```
# Combining all the above students
from tqdm import tqdm
```



```

preprocessed_reviews = []
# tqdm is for printing the status bar
for sentence in tqdm(final['Text'].values):
    sentence = re.sub(r"http\S+", "", sentence)
    sentence = BeautifulSoup(sentence, 'lxml').get_text()
    sentence = decontracted(sentence)
    sentence = re.sub("\S*\d\S*", "", sentence).strip()
    sentence = re.sub('[^A-Za-z]+', ' ', sentence)
    # https://gist.github.com/sebleier/554280
    sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentence.strip())

```

100%|██████████| 364171/364171 [02:40<00:00, 2276.04it/s]

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

'great ingredients although chicken rather chicken broth thing not think belongs canola oil canola rapeseed not someting dog would ever find nature find rapeseed nature eat would poison today food industries convinced masses canola oil safe even better oil olive virgin coconut facts though say otherwise late poisonous figured way fix still like could better'

## [3.2] Preprocessing Review Summary

In [24]:

```

## Summary preprocessing
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentence in tqdm(final['Summary'].values):
    sentence = re.sub(r"http\S+", "", sentence)
    sentence = BeautifulSoup(sentence, 'lxml').get_text()
    sentence = decontracted(sentence)
    sentence = re.sub("\S*\d\S*", "", sentence).strip()
    sentence = re.sub('[^A-Za-z]+', ' ', sentence)
    # https://gist.github.com/sebleier/554280
    sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwords)
    preprocessed_summary.append(sentence.strip())

```

100%|██████████| 364171/364171 [01:44<00:00, 3479.00it/s]

In [23]:

```
final['Cleaned_text'] = preprocessed_reviews
```

In [24]:

```

### Sort data according to time series
final.sort_values('Time', inplace=True)

```

In [40]:

```

### Taking 100k samples
final_50k = final.sample(n=50000)
final_50k.head(3)

```

Out[40]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Si
126734	137513	B001E530J8	AGXUKOSN9VO99	remiap48	3	3	1	1339891200	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
287057	310968	B00064VQNU	A26LO3KVCC5ZGQ	Angela Katsavavakis	3	4	1	1225929600	to
191661	207803	B0086HGVXY	AY9UGFEEGOM8N	R. Slade "asteviefan"	0	0	1	1344297600	M e

In [41]:

```
# Removing the score column as it is not needed.
sample_data = final_50k.drop('Score', axis = 1)
sample_data.head(2)
```

Out[41]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Time	Summary
126734	137513	B001E530J8	AGXUKOSN9VO99	remlap48	3	3	1339891200	Great coffee, but.. HOW MUCH??
287057	310968	B00064VQNU	A26LO3KVCC5ZGQ	Angela Katsavavakis	3	4	1225929600	Reasons to buy this olive oil

In [42]:

```
final_5k = sample_data.sample(n=5000)
```

In [43]:

```
x = sample_data['Cleaned_text']
```

## [4] Featurization

### [4.1] BAG OF WORDS

In [44]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
bow = count_vect.fit_transform(sample_data['Cleaned_text'].values)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

#x_test_bow = count_vect.transform(x_test)
print("the type of count vectorizer ",type(bow))
print("the shape of out text BOW vectorizer ",bow.get_shape())
print("the number of unique words ", bow.get_shape()[1])
```

some feature names ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaa', 'aaaaah', 'aaaah', 'aaah', 'aaahhh', 'aachener']

```
=====
the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer (50000, 43419)
the number of unique words 43419
```

## [4.2] Bi-Grams and n-Grams.

In [30]:

```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.CountVectorizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(x)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_shape()[1])
```

the type of count vectorizer <class 'scipy.sparse.csr.csr\_matrix'>  
the shape of out text BOW vectorizer (100000, 5000)  
the number of unique words including both unigrams and bigrams 5000

## [4.3] TF-IDF

In [45]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tfidf = tf_idf_vect.fit_transform(x)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

#x_test_tfidf = tf_idf_vect.transform(x_test)
print("the type of count vectorizer ",type(tfidf))
print("the shape of out text TFIDF vectorizer ",tfidf.get_shape())
print("the number of unique words including both unigrams and bigrams ", tfidf.get_shape()[1])
```

some sample features(unique words in the corpus) ['ability', 'able', 'able buy', 'able drink', 'able eat', 'able enjoy', 'able find', 'able get', 'able give', 'able go']  
=====

the type of count vectorizer <class 'scipy.sparse.csr.csr\_matrix'>  
the shape of out text TFIDF vectorizer (50000, 29322)  
the number of unique words including both unigrams and bigrams 29322

## [4.4] Word2Vec

In [46]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentence=[]
for sentence in x:
    list_of_sentence.append(sentence.split())
```

In [47]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.
```

```
# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these variable according to your need

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 occurred at least 5 times
    w2v_model=Word2Vec(list_of_sentence,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-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have google's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")
```

```
[('wonderful', 0.8433031439781189), ('good', 0.8321810960769653), ('excellent',
0.8308950662612915), ('fantastic', 0.7974873781204224), ('perfect', 0.7710416316986084),
('terrific', 0.7648207545280457), ('awesome', 0.7552918791770935), ('amazing',
0.7248703837394714), ('nice', 0.6933922171592712), ('fabulous', 0.6891815066337585)]
=====
[('nastiest', 0.7648350596427917), ('greatest', 0.7552139759063721), ('best', 0.7541465759277344),
('closest', 0.6806847453117371), ('nobody', 0.6259416341781616), ('tastiest', 0.6219135522842407),
('healthiest', 0.6215877532958984), ('smoothest', 0.6164917945861816), ('surpasses',
0.6089249849319458), ('disgusting', 0.5958737134933472)]
```

In [48]:

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

```
number of words that occurred minimum 5 times 13807
sample words ['back', 'reviews', 'coffee', 'full', 'bodied', 'smooth', 'great', 'taste',
'seriously', 'cans', 'know', 'fool', 'money', 'soon', 'easily', 'found', 'much', 'less',
'example', 'purchased', 'yuban', 'last', 'week', 'well', 'known', 'discount', 'clearance',
'store', 'not', 'say', 'name', 'big', 'sells', 'lots', 'stuff', 'sorry', 'digress', 'anyway', 'point',
'shop', 'around', 'buy', 'family', 'owned', 'operated', 'made', 'following', 'traditional',
'authentic', 'italian']
```

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

### [4.4.1.1] Avg W2v

In [49]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentence): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
    # 300 if you use google's w2v
    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)
```

```

sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))

```

100%|██████████| 50000/50000 [02:05<00:00, 397.89it/s]

50000  
50

#### [4.4.1.2] TFIDF weighted W2v

In [50]:

```

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer(min_df=10, max_features=500)
tf_idf_matrix = model.fit_transform(x)
# 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 [51]:

```

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentence): # 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 corpus
            # sent.count(word) = tf value of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1

```

100%|██████████| 50000/50000 [02:25<00:00, 343.55it/s]

## [5] Assignment 10: K-Means, Agglomerative & DBSCAN Clustering

### 1. Apply K-means Clustering on these feature sets:

- **SET 1:** Review text, preprocessed one converted into vectors using (BOW)
- **SET 2:** Review text, preprocessed one converted into vectors using (TFIDF)
- **SET 3:** Review text, preprocessed one converted into vectors using (AVG W2v)
- **SET 4:** Review text, preprocessed one converted into vectors using (TFIDF W2v)
- Find the best 'k' using the elbow-knee method (plot k vs inertia\_)
- Once after you find the k clusters, plot the word cloud per each cluster so that at a single go we can analyze the words in a cluster.

### 2. Apply Agglomerative Clustering on these feature sets:

- **SET 3:** Review text, preprocessed one converted into vectors using (AVG W2v)
- **SET 4:** Review text, preprocessed one converted into vectors using (TFIDF W2v)
- Apply agglomerative algorithm and try a different number of clusters like 2,5 etc.

- Same as that of K-means, plot word clouds for each cluster and summarize in your own words what that cluster is representing.
- You can take around 5000 reviews or so (as this is very computationally expensive one)

### 3. Apply DBSCAN Clustering on these feature sets:

- **SET 3:** Review text, preprocessed one converted into vectors using (AVG W2v)
- **SET 4:** Review text, preprocessed one converted into vectors using (TFIDF W2v)
- Find the best 'Eps' using the [elbow-knee method](#).
- Same as before, plot word clouds for each cluster and summarize in your own words what that cluster is representing.
- You can take around 5000 reviews for this as well.

## [5.1] K-Means Clustering

### [5.1.1] Applying K-Means Clustering on BOW, **SET 1**

In [52]:

```
X = final_50k['Cleaned_text'].values
```

In [53]:

```
count_vect = CountVectorizer()
bow = count_vect.fit_transform(X)
```

In [54]:

```
list_of_sent = []
for i in X:
    sent = []
    for word in i.split():
        sent.append(word)
    list_of_sent.append(sent)
```

In [55]:

```
from sklearn.cluster import KMeans
```

In [42]:

```
num_clus = [x for x in range(3,11)]
num_clus
```

Out[42]:

```
[3, 4, 5, 6, 7, 8, 9, 10]
```

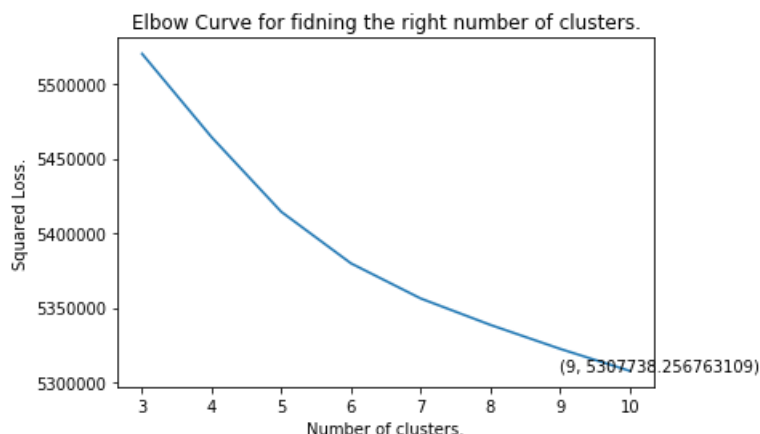
#### #### elbow method

In [43]:

```
### Hyperparameter tuning for k clusters
squared_errors = []
for cluster in num_clus:
    kmeans = KMeans(n_clusters = cluster, n_jobs = -1)
    kmeans = kmeans.fit(bow)
    squared_errors.append(kmeans.inertia_)

optimal_clusters = np.argmin(squared_errors) + 2
plt.plot(num_clus, squared_errors)
plt.title("Elbow Curve for finding the right number of clusters.")
plt.xlabel("Number of clusters.")
plt.ylabel("Squared Loss.")
xy = (optimal_clusters, min(squared_errors))
plt.annotate('(%s, %s)' % xy, xy = xy, textcoords='data')
plt.show()
```

```
print ("The optimal number of clusters obtained is - ", optimal_clusters)
print ("The loss for optimal cluster is - ", min(squared_errors))
```



The optimal number of clusters obtained is - 9  
The loss for optimal cluster is - 5307738.256763109

In [44]:

```
### Train model with optimal clusters
```

In [56]:

```
optimal_k = KMeans(n_clusters = 9)
p = optimal_k.fit_predict(bow)
```

In [57]:

```
### append a same label cluster in index
index_0 = []
index_1 = []
index_2 = []
index_3 = []
index_4 = []
index_5 = []
index_6 = []
index_7 = []
index_8 = []

for i in range(len(p)):
    if p[i] == 0:
        index_0.append(i)
for i in range(len(p)):
    if p[i] == 1:
        index_1.append(i)
for i in range(len(p)):
    if p[i] == 2:
        index_2.append(i)
for i in range(len(p)):
    if p[i] == 3:
        index_3.append(i)
for i in range(len(p)):
    if p[i] == 4:
        index_4.append(i)
for i in range(len(p)):
    if p[i] == 5:
        index_5.append(i)
for i in range(len(p)):
    if p[i] == 6:
        index_6.append(i)
for i in range(len(p)):
    if p[i] == 7:
        index_7.append(i)
for i in range(len(p)):
    if p[i] == 8:
        index_8.append(i)
```

```
text_0 = []
text_1 = []
text_2 = []
text_3 = []
text_4 = []
text_5 = []
text_6 = []
text_7 = []
text_8 = []

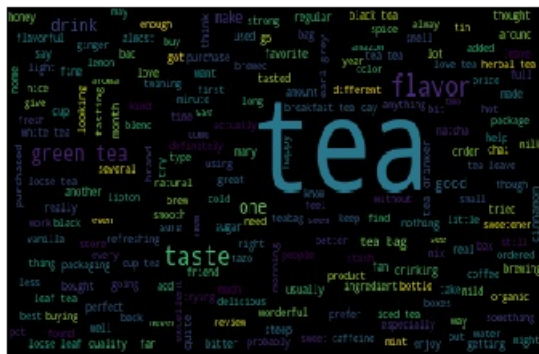
for i in range(len(index_0)):
    text_0.append(list_of_sent[index_0[i]])
for i in range(len(index_1)):
    text_1.append(list_of_sent[index_1[i]])
for i in range(len(index_2)):
    text_2.append(list_of_sent[index_2[i]])
for i in range(len(index_3)):
    text_3.append(list_of_sent[index_3[i]])
for i in range(len(index_4)):
    text_4.append(list_of_sent[index_4[i]])
for i in range(len(index_5)):
    text_5.append(list_of_sent[index_5[i]])
for i in range(len(index_6)):
    text_6.append(list_of_sent[index_6[i]])
for i in range(len(index_7)):
    text_7.append(list_of_sent[index_7[i]])
for i in range(len(index_8)):
    text_8.append(list_of_sent[index_8[i]])
```

## In [59]:

```
## Create a wordcloud of cluster 1
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_1)):
    for i in range(len(text_1[j])):
```



```
t_b = t_b + text_1[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [61]:

```
## Create a wordcloud of cluster 2
from wordcloud import WordCloud
from matplotlib.pyplot import figure

t_b = ''
for j in range(len(text_2)):
    for i in range(len(text_2[j])):
        t_b = t_b + text_2[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [51]:

```
## Create a wordcloud of cluster 3
from wordcloud import WordCloud
from matplotlib.pyplot import figure

t_b = ''
for j in range(len(text_3)):
    for i in range(len(text_3[j])):
        t_b = t_b + text_3[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```





In [52]:

```
## Cleave a wordcloud of cluster 4
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_4)):
    for i in range(len(text_4[j])):
        t_b = t_b + text_4[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [53]:

```
## Cleave a wordcloud of cluster 5
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_5)):
    for i in range(len(text_5[j])):
        t_b = t_b + text_5[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [54]:

```
## Cleave a wordcloud of cluster 6
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_6)):
    for i in range(len(text_6[j])):
        t_b = t_b + text_6[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [55]:

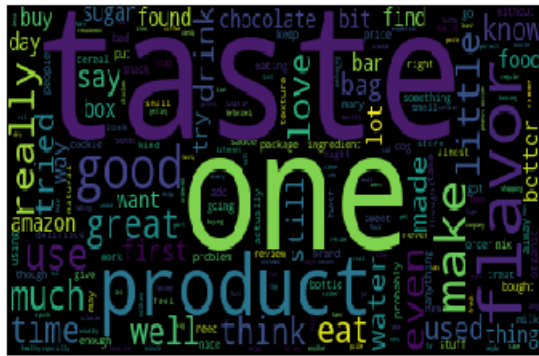
```
## Cleave a wordcloud of cluster 7
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_7)):
    for i in range(len(text_7[j])):
        t_b = t_b + text_7[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [62]:

```
## Cleave a wordcloud of cluster 8
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_8)):
    for i in range(len(text_8[j])):
        t_b = t_b + text_8[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```

```
plt.axis('off')
plt.show()
```



Observation: K means with BOW contain words are of bevarages category i.e tea, coffee, etc.

### [5.1.3] Applying K-Means Clustering on TFIDF, SET 2

In [57]:

```
num_clus = [x for x in range(3,11)]
num_clus
```

Out[57]:

```
[3, 4, 5, 6, 7, 8, 9, 10]
```

In [58]:

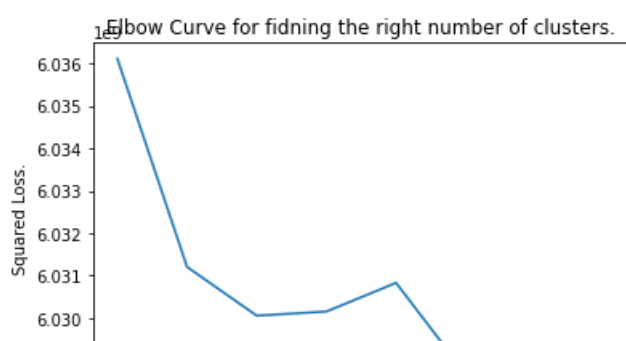
```
tfidf_vect = TfidfVectorizer()
tfidf_train = tfidf_vect.fit_transform(X)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean = False)
X_tr = sc.fit_transform(tfidf_train)
```

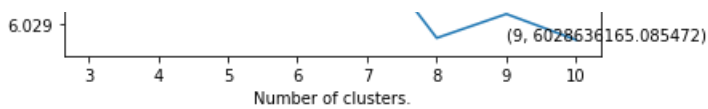
In [59]:

```
squared_errors = []
for cluster in num_clus:
    kmeans = KMeans(n_clusters = cluster, n_jobs = -1)
    kmeans = kmeans.fit(X_tr)
    squared_errors.append(kmeans.inertia_)

optimal_clusters = np.argmin(squared_errors) + 2
plt.plot(num_clus, squared_errors)
plt.title("Elbow Curve for fidning the right number of clusters.")
plt.xlabel("Number of clusters.")
plt.ylabel("Squared Loss.")
xy = (optimal_clusters, min(squared_errors))
plt.annotate('(%s, %s)' % xy, xy = xy, textcoords='data')
plt.show()

print ("The optimal number of clusters obtained is - ", optimal_clusters)
print ("The loss for optimal cluster is - ", min(squared_errors))
```





The optimal number of clusters obtained is - 9  
 The loss for optimal cluster is - 6028636165.085472

In [60]:

```
### Train model with optimal clusters
opt_k = KMeans(n_clusters = 7)
p = optimal_k.fit_predict(tfidf_train)
```

In [61]:

```
### append a same label cluster in index
index_0 = []
index_1 = []
index_2 = []
index_3 = []
index_4 = []
index_5 = []
index_6 = []
index_7 = []
index_8 = []

for i in range(len(p)):
    if p[i] == 0:
        index_0.append(i)
for i in range(len(p)):
    if p[i] == 1:
        index_1.append(i)
for i in range(len(p)):
    if p[i] == 2:
        index_2.append(i)
for i in range(len(p)):
    if p[i] == 3:
        index_3.append(i)
for i in range(len(p)):
    if p[i] == 4:
        index_4.append(i)
for i in range(len(p)):
    if p[i] == 5:
        index_5.append(i)
for i in range(len(p)):
    if p[i] == 6:
        index_6.append(i)
for i in range(len(p)):
    if p[i] == 7:
        index_7.append(i)
for i in range(len(p)):
    if p[i] == 8:
        index_8.append(i)
```

In [62]:

```
text_0 = []
text_1 = []
text_2 = []
text_3 = []
text_4 = []
text_5 = []
text_6 = []
text_7 = []
text_8 = []

for i in range(len(index_0)):
    text_0.append(list_of_sent[index_0[i]])
for i in range(len(index_1)):
    text_1.append(list_of_sent[index_1[i]])
for i in range(len(index_2)):
    text_2.append(list_of_sent[index_2[i]])
for i in range(len(index_3)):
    text_3.append(list_of_sent[index_3[i]])
```

```

text_3.append(list_of_sent[index_3[i]])
for i in range(len(index_4)):
    text_4.append(list_of_sent[index_4[i]])
for i in range(len(index_5)):
    text_5.append(list_of_sent[index_5[i]])
for i in range(len(index_6)):
    text_6.append(list_of_sent[index_6[i]])
for i in range(len(index_7)):
    text_7.append(list_of_sent[index_7[i]])
for i in range(len(index_8)):
    text_8.append(list_of_sent[index_8[i]])

```

#### [5.1.4] Wordclouds of clusters obtained after applying k-means on TFIDF SET 2

In [63]:

```

## Create a wordcloud of cluster 0
from wordcloud import WordCloud
from matplotlib.pyplot import figure

t_b = ''
for j in range(len(text_0)):
    for i in range(len(text_0[j])):
        t_b = t_b + text_0[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

```



In [64]:

```

## Create a wordcloud of cluster 1
from wordcloud import WordCloud
from matplotlib.pyplot import figure

t_b = ''
for j in range(len(text_1)):
    for i in range(len(text_1[j])):
        t_b = t_b + text_1[j][i] + ' '
#print(t_b)

word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

```







```

#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

```

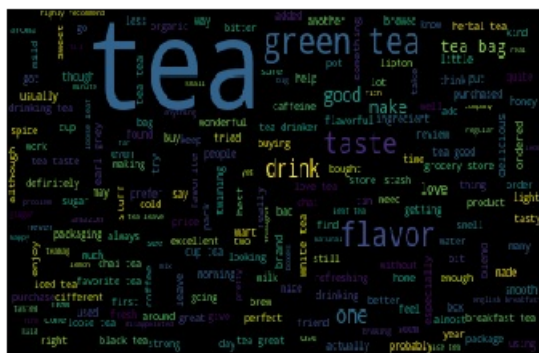


In [68]:

```

## Cleave a wordcloud of cluster 5
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_5)):
    for i in range(len(text_5[j])):
        t_b = t_b + text_5[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

```



In [69]:

```

## Cleave a wordcloud of cluster 6
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_6)):
    for i in range(len(text_6[j])):
        t_b = t_b + text_6[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

```







```
In [ ]:
```

Observation:

K means **with** TFIDF contain words are of bevarages catogory **and** Chocolates also.

### [5.1.5] Applying K-Means Clustering on AVG W2V, SET 3

```
In [72]:
```

```
sent_vectors = np.array(sent_vectors)
sent_vectors = np.nan_to_num(sent_vectors)
sent_vectors.shape
```

```
Out[72]:
```

```
(100000, 50)
```

```
In [73]:
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
w2v = sc.fit_transform(sent_vectors)
```

```
In [74]:
```

```
num_clus = [x for x in range(3,11)]
num_clus
```

```
Out[74]:
```

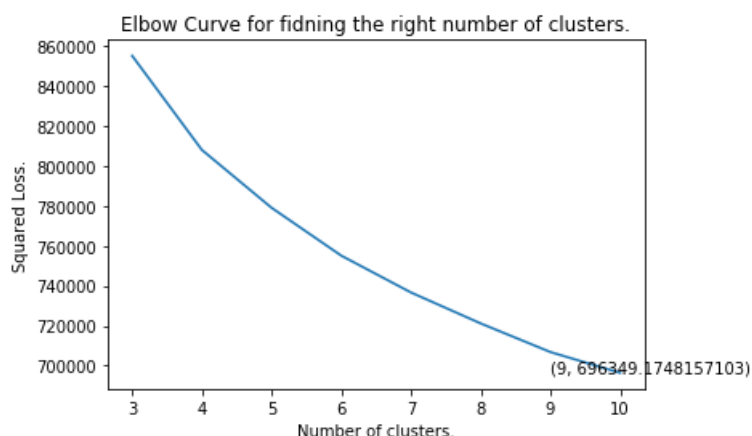
```
[3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [75]:
```

```
from sklearn.cluster import KMeans
squared_errors = []
for cluster in num_clus:
    kmeans = KMeans(n_clusters = cluster, n_jobs = -1)
    kmeans = kmeans.fit(sent_vectors)
    squared_errors.append(kmeans.inertia_)

optimal_clusters = np.argmin(squared_errors) + 2
plt.plot(num_clus, squared_errors)
plt.title("Elbow Curve for fidning the right number of clusters.")
plt.xlabel("Number of clusters.")
plt.ylabel("Squared Loss.")
xy = (optimal_clusters, min(squared_errors))
plt.annotate('%s, %s' % xy, xy = xy, textcoords='data')
plt.show()

print ("The optimal number of clusters obtained is - ", optimal_clusters)
print ("The loss for optimal cluster is - ", min(squared_errors))
```



The optimal number of clusters obtained is - 9

The optimal number of clusters obtained is - 9  
The loss for optimal cluster is - 696349.1748157103

In [76]:

```
### Train model with optimal clusters
from sklearn.preprocessing import normalize
model = KMeans(n_clusters = optimal_clusters, n_jobs = -1)
p = model.fit_predict(normalize(sent_vectors))
```

In [77]:

```
### append a same label cluster in index
index_0 = []
index_1 = []
index_2 = []
index_3 = []
index_4 = []
index_5 = []
index_6 = []
index_7 = []
index_8 = []

for i in range(len(p)):
    if p[i] == 0:
        index_0.append(i)
for i in range(len(p)):
    if p[i] == 1:
        index_1.append(i)
for i in range(len(p)):
    if p[i] == 2:
        index_2.append(i)
for i in range(len(p)):
    if p[i] == 3:
        index_3.append(i)
for i in range(len(p)):
    if p[i] == 4:
        index_4.append(i)
for i in range(len(p)):
    if p[i] == 5:
        index_5.append(i)
for i in range(len(p)):
    if p[i] == 6:
        index_6.append(i)
for i in range(len(p)):
    if p[i] == 7:
        index_7.append(i)
for i in range(len(p)):
    if p[i] == 8:
        index_8.append(i)
```

In [78]:

```
text_0 = []
text_1 = []
text_2 = []
text_3 = []
text_4 = []
text_5 = []
text_6 = []
text_7 = []
text_8 = []

for i in range(len(index_0)):
    text_0.append(list_of_sent[index_0[i]])
for i in range(len(index_1)):
    text_1.append(list_of_sent[index_1[i]])
for i in range(len(index_2)):
    text_2.append(list_of_sent[index_2[i]])
for i in range(len(index_3)):
    text_3.append(list_of_sent[index_3[i]])
for i in range(len(index_4)):
    text_4.append(list_of_sent[index_4[i]])
for i in range(len(index_5)):
    text_5.append(list_of_sent[index_5[i]])
```

**[5.1.6] Wordclouds of clusters obtained after applying k-means on AVG W2V SET 3**

```

## Create a wordcloud of cluster 0
from wordcloud import WordCloud
from matplotlib.pyplot import figure

t_b = ''
for j in range(len(text_0)):
    for i in range(len(text_0[j])):
        t_b = t_b + text_0[j][i] + ' '
#print(t_b)

word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

```



```

## Create a wordcloud of cluster 1
from wordcloud import WordCloud
from matplotlib.pyplot import figure

t_b = ''
for j in range(len(text_1)):
    for i in range(len(text_1[j])):
        t_b = t_b + text_1[j][i] + ' '
#print(t_b)

word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

```



In [81]:

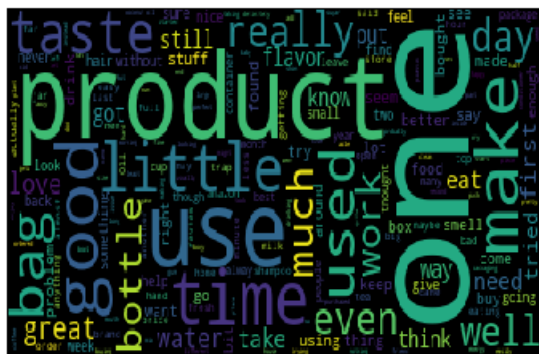


```
plt.show()
```



In [84]:

```
## Cleave a wordcloud of cluster 5
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_5)):
    for i in range(len(text_5[j])):
        t_b = t_b + text_5[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud, aspect='auto')
plt.axis('off')
plt.show()
```



In [85]:

```
## Cleave a wordcloud of cluster 6
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_6)):
    for i in range(len(text_6[j])):
        t_b = t_b + text_6[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud, aspect='auto')
plt.axis('off')
plt.show()
```





In [86]:

```
## Cleave a wordcloud of cluster 7
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_7)):
    for i in range(len(text_7[j])):
        t_b = t_b + text_7[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud, aspect='auto')
plt.axis('off')
plt.show()
```



In [87]:

```
## Cleave a wordcloud of cluster 8
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_8)):
    for i in range(len(text_8[j])):
        t_b = t_b + text_8[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud, aspect='auto')
plt.axis('off')
plt.show()
```



Observation: K means with AVG-W2V contain words are of grocery, bevarages , cat food and dog food catogory.

### [5.1.7] Applying K-Means Clustering on TFIDF W2V, SET 4

In [88]:

In [88]:

```
tfidf_sent_vectors = np.array(tfidf_sent_vectors)
tfidf_sent_vectors = np.nan_to_num(tfidf_sent_vectors)
tfidf_sent_vectors.shape
```

Out[88]:

(100000, 50)

In [89]:

```
sc = StandardScaler()
tfidf_w2v = sc.fit_transform(tfidf_sent_vectors)
```

In [90]:

```
num_clus = [x for x in range(3,11)]
num_clus
```

Out[90]:

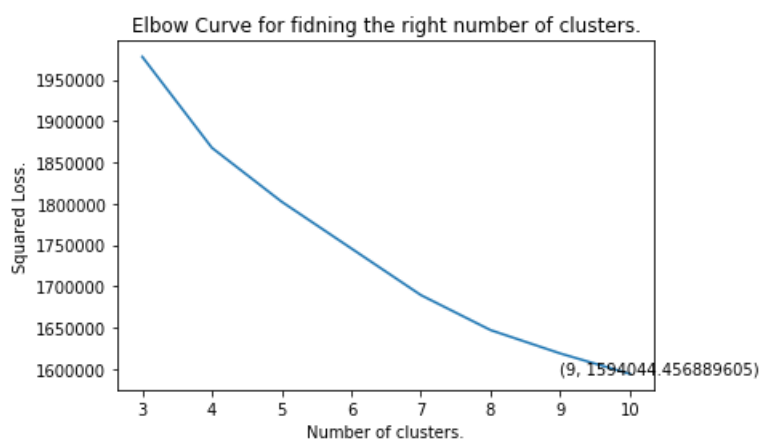
[3, 4, 5, 6, 7, 8, 9, 10]

In [91]:

```
squared_errors = []
for cluster in num_clus:
    kmeans = KMeans(n_clusters = cluster, n_jobs = -1)
    kmeans = kmeans.fit(tfidf_sent_vectors)
    squared_errors.append(kmeans.inertia_)

optimal_clusters = np.argmin(squared_errors) + 2
plt.plot(num_clus, squared_errors)
plt.title("Elbow Curve for finding the right number of clusters.")
plt.xlabel("Number of clusters.")
plt.ylabel("Squared Loss.")
xy = (optimal_clusters, min(squared_errors))
plt.annotate('%s, %s' % xy, xy = xy, textcoords='data')
plt.show()

print ("The optimal number of clusters obtained is - ", optimal_clusters)
print ("The loss for optimal cluster is - ", min(squared_errors))
```



The optimal number of clusters obtained is - 9  
The loss for optimal cluster is - 1594044.456889605

In [92]:

```
### Train model with optimal clusters
from sklearn.preprocessing import normalize
model = KMeans(n_clusters = optimal_clusters, n_jobs = -1)
p = model.fit_predict(normalize(tfidf_sent_vectors))
```



In [93]:

```
### append a same label cluster in index
index_0 = []
index_1 = []
index_2 = []
index_3 = []
index_4 = []
index_5 = []
index_6 = []
index_7 = []
index_8 = []

for i in range(len(p)):
    if p[i] == 0:
        index_0.append(i)
for i in range(len(p)):
    if p[i] == 1:
        index_1.append(i)
for i in range(len(p)):
    if p[i] == 2:
        index_2.append(i)
for i in range(len(p)):
    if p[i] == 3:
        index_3.append(i)
for i in range(len(p)):
    if p[i] == 4:
        index_4.append(i)
for i in range(len(p)):
    if p[i] == 5:
        index_5.append(i)
for i in range(len(p)):
    if p[i] == 6:
        index_6.append(i)
for i in range(len(p)):
    if p[i] == 7:
        index_7.append(i)
for i in range(len(p)):
    if p[i] == 8:
        index_8.append(i)
```

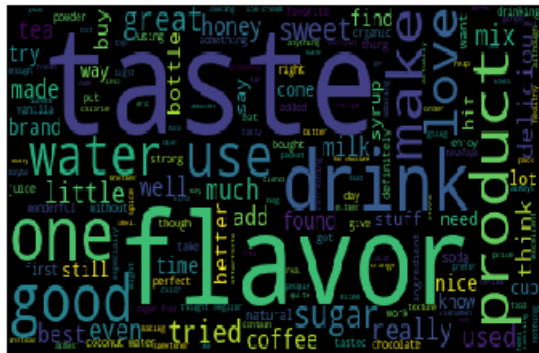
In [94]:

```
text_0 = []
text_1 = []
text_2 = []
text_3 = []
text_4 = []
text_5 = []
text_6 = []
text_7 = []
text_8 = []

for i in range(len(index_0)):
    text_0.append(list_of_sent[index_0[i]])
for i in range(len(index_1)):
    text_1.append(list_of_sent[index_1[i]])
for i in range(len(index_2)):
    text_2.append(list_of_sent[index_2[i]])
for i in range(len(index_3)):
    text_3.append(list_of_sent[index_3[i]])
for i in range(len(index_4)):
    text_4.append(list_of_sent[index_4[i]])
for i in range(len(index_5)):
    text_5.append(list_of_sent[index_5[i]])
for i in range(len(index_6)):
    text_6.append(list_of_sent[index_6[i]])
for i in range(len(index_7)):
    text_7.append(list_of_sent[index_7[i]])
for i in range(len(index_8)):
    text_8.append(list_of_sent[index_8[i]])
```

In [95]:

```
## Cleave a wordcloud of cluster 0
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_0)):
    for i in range(len(text_0[j])):
        t_b = t_b + text_0[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [96]:

```
## Cleave a wordcloud of cluster 1
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_1)):
    for i in range(len(text_1[j])):
        t_b = t_b + text_1[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [97]:

```
## Cleave a wordcloud of cluster 2
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_2)):
    for i in range(len(text_2[j])):
        t_b = t_b + text_2[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



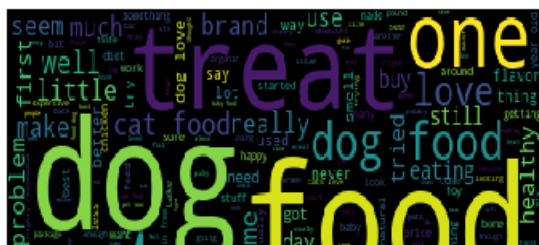
```
In [98]:
```

```
## Cleave a wordcloud of Cluster 3
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_3)):
    for i in range(len(text_3[j])):
        t_b = t_b + text_3[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



```
In [99]:
```

```
## Cleave a wordcloud of Cluster 4
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_4)):
    for i in range(len(text_4[j])):
        t_b = t_b + text_4[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```





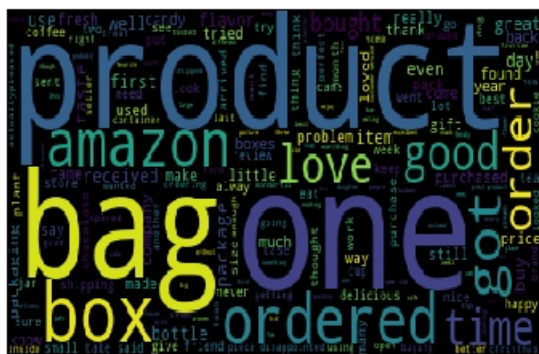
In [100]:

```
## Cleave a wordcloud of cluster 5
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_5)):
    for i in range(len(text_5[j])):
        t_b = t_b + text_5[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [101]:

```
## Cleave a wordcloud of cluster 6
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_6)):
    for i in range(len(text_6[j])):
        t_b = t_b + text_6[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [102]:

```
## Cleave a wordcloud of cluster 8
from wordcloud import WordCloud
from matplotlib.pyplot import figure
```

```

t_b = ''
for j in range(len(text_8)):
    for i in range(len(text_8[j])):
        t_b = t_b + text_8[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

```



Observation: K means with TFIDF-W2V contain some words are of sugar,honey like sweet food catogory etc.

## [5.2] Agglomerative Clustering

### [5.2.1] Applying Agglomerative Clustering on AVG W2V, SET 3

In [63]:

```
x = final_5k['Cleaned_text']
```

In [64]:

```

# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentence=[]
for sentence in x:
    list_of_sentence.append(sentence.split())

```

In [65]:

```

# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.

# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these variable according to your need

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_sentence,min_count=5,size=50,workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('great'))

```

```

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-negative300.bin', binary=True)
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")

```

```

[('pretty', 0.9945371150970459), ('wonderful', 0.9936685562133789), ('real', 0.9932069182395935),
('cinnamon', 0.9930114150047302), ('light', 0.9929517507553101), ('tasting', 0.9929429888725281),
('flavorful', 0.9929109215736389), ('smoother', 0.9928531646728516), ('overpowering',
0.9928467869758606), ('refreshing', 0.9928240776062012)]
=====
[('hands', 0.9993862509727478), ('name', 0.9992755651473999), ('cafe', 0.9992227554321289),
('experience', 0.9992014765739441), ('hate', 0.999191403388977), ('simply', 0.9991807341575623), (
'fantastic', 0.9991796612739563), ('oh', 0.9991793036460876), ('favorites', 0.9991747140884399), (
'either', 0.9991582632064819)]

```

In [66]:

```

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

```

```

number of words that occurred minimum 5 times 4142
sample words ['found', 'brick', 'store', 'much', 'less', 'usually', 'find', 'amazon', 'time',
'lets', 'plus', 'first', 'posted', 'message', 'suspect', 'listed', 'could', 'purchase', 'beware',
'product', 'not', 'pictured', 'shown', 'pill', 'pockets', 'capsules', 'oz', 'bag', 'delivered', 's
maller', 'tablets', 'concerned', 'quality', 'ingredients', 'dog', 'food', 'shopping', 'around', 'y
es', 'chihuahua', 'picky', 'spoiled', 'actually', 'boiling', 'chicken', 'breast', 'mixing', 'dry',
'would', 'eat']

```

In [67]:

```

# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentence): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
    to 300 if you use google's w2v
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))

```

```

100%|██████████| 5000/5000 [00:06<00:00, 738.45it/s]

```

```

5000
50

```

In [68]:

```

sent_vectors = np.array(sent_vectors)
sent_vectors = np.nan_to_num(sent_vectors)
sent_vectors.shape

```

Out[68]:

```

(5000, 50)

```

In [69]:

```
from sklearn.cluster import AgglomerativeClustering
```

In [70]:

```
list_of_sent = []
for i in X:
    sent = []
    for word in i.split():
        sent.append(word)
    list_of_sent.append(sent)
```

In [71]:

```
np.where(np.isnan(sent_vectors))
```

Out[71]:

```
(array([], dtype=int64), array([], dtype=int64))
```

In [72]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
w2v = sc.fit_transform(sent_vectors)
```

In [73]:

```
from sklearn.cluster import AgglomerativeClustering
agg = AgglomerativeClustering(n_clusters = 5)
p = agg.fit_predict(w2v)
```

In [74]:

```
### append a same label cluster in index
index_0 = []
index_1 = []
index_2 = []
index_3 = []
index_4 = []

for i in range(len(p)):
    if p[i] == 0:
        index_0.append(i)
for i in range(len(p)):
    if p[i] == 1:
        index_1.append(i)
for i in range(len(p)):
    if p[i] == 2:
        index_2.append(i)
for i in range(len(p)):
    if p[i] == 3:
        index_3.append(i)
for i in range(len(p)):
    if p[i] == 4:
        index_4.append(i)
```

In [75]:

```
text_0 = []
text_1 = []
text_2 = []
text_3 = []
text_4 = []

for i in range(len(index_0)):
    text_0.append(list_of_sent[index_0[i]])
for i in range(len(index_1)):
    text_1.append(list_of_sent[index_1[i]])
```

### [5.2.2] Wordclouds of clusters obtained after applying Agglomerative Clustering on AVG W2V SET 3

```
## Create a wordcloud of cluster 0
from wordcloud import WordCloud
from matplotlib.pyplot import figure

t_b = ''
for j in range(len(text_0)):
    for i in range(len(text_0[j])):
        t_b = t_b + text_0[j][i] + ' '
#print(t_b)

word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



```
## Create a wordcloud of cluster 1
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_1)):
    for i in range(len(text_1[j])):
        t_b = t_b + text_1[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```





In [78]:

```
## Create a wordcloud of cluster 2
from wordcloud import WordCloud
from matplotlib.pyplot import figure

t_b = ''
for j in range(len(text_2)):
    for i in range(len(text_2[j])):
        t_b = t_b + text_2[j][i] + ' '
#print(t_b)

word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [79]:

```

## Create a wordcloud of cluster 3
from wordcloud import WordCloud
from matplotlib.pyplot import figure

t_b = ''
for j in range(len(text_3)):
    for i in range(len(text_3[j])):
        t_b = t_b + text_3[j][i] + ' '
#print(t_b)

word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

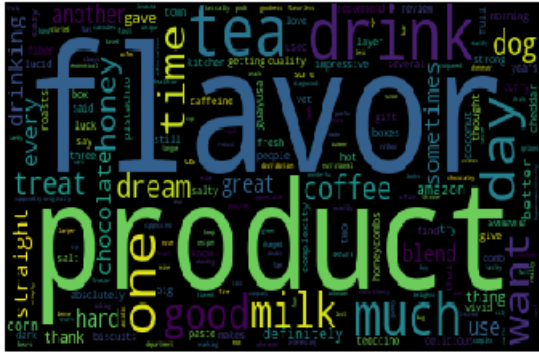
```



In [80]:

```
## Cleave a wordcloud of cluster 4
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_4)):
    for i in range(len(text_4[j])):
        t_b = t_b + text_4[j][i] + ' '
#print(t_b)
word cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
```

```
plt.imshow(word_cloud, aspect='auto')
plt.axis('off')
plt.show()
```



Observation: Agglomerative with AVG-W2V contain words are of bevarages catogory i.e tea, coffee, milk etc.### Apply agglomerative algorithm and try number of clusters like 2.

In [81]:

```
from sklearn.cluster import AgglomerativeClustering
agg = AgglomerativeClustering(n_clusters = 2)
p = agg.fit_predict(w2v)
```

In [82]:

```
### append a same label cluster in index
index_0 = []
index_1 = []

for i in range(len(p)):
    if p[i] == 0:
        index_0.append(i)
for i in range(len(p)):
    if p[i] == 1:
        index_1.append(i)
```

In [ ]:

### ### Wordclouds of clusters obtained after applying Agglomerative Clustering on AVG W2V

In [83]:

```
## Create a wordcloud of cluster 4
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_0)):
    for i in range(len(text_0[j])):
        t_b = t_b + text_0[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```





In [84]:

```
## Cleave a wordcloud of cluster 4
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_1)):
    for i in range(len(text_1[j])):
        t_b = t_b + text_1[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud, aspect='auto')
plt.axis('off')
plt.show()
```



Observation: Agglomerative with AVG-W2V contain words are of beverages category i.e tea, coffee, etc.

### [5.2.3] Applying Agglomerative Clustering on TFIDF W2V, SET 4

In [85]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(x)
# 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 [86]:

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentence): # 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 corpus
            # sent.count(word) = tf value of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

100%|██████████| 5000/5000 [01:16<00:00, 68.78it/s]

In [87]:

```
tfidf_sent_vectors = np.array(tfidf_sent_vectors)
tfidf_sent_vectors = np.nan_to_num(tfidf_sent_vectors)
tfidf_sent_vectors.shape
```

Out[87]:

(5000, 50)

In [88]:

```
np.where(np.isnan(tfidf_sent_vectors))
```

Out[88]:

(array([], dtype=int64), array([], dtype=int64))

In [89]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
tf_w2v = sc.fit_transform(tfidf_sent_vectors)
```

In [90]:

```
from sklearn.cluster import AgglomerativeClustering
agg = AgglomerativeClustering(n_clusters = 5)
p = agg.fit_predict(tf_w2v)
```

In [115]:

```
### append a same label cluster in index
index_0 = []
index_1 = []
index_2 = []
index_3 = []
index_4 = []

for i in range(len(p)):
    if p[i] == 0:
        index_0.append(i)
for i in range(len(p)):
    if p[i] == 1:
        index_1.append(i)
for i in range(len(p)):
    if p[i] == 2:
        index_2.append(i)
for i in range(len(p)):
    if p[i] == 3:
        index_3.append(i)
for i in range(len(p)):
    if p[i] == 4:
        index_4.append(i)
```

In [116]:

```
text_0 = []
text_1 = []
text_2 = []
text_3 = []
text_4 = []

for i in range(len(index_0)):
    text_0.append(list_of_sent[index_0[i]])
for i in range(len(index_1)):
    text_1.append(list_of_sent[index_1[i]])
```

```
for i in range(len(index_2)):
    text_2.append(list_of_sent[index_2[i]])
for i in range(len(index_3)):
    text_3.append(list_of_sent[index_3[i]])
for i in range(len(index_4)):
    text_4.append(list_of_sent[index_4[i]])
```

**[5.2.4] Wordclouds of clusters obtained after applying Agglomerative Clustering on TFIDF W2V**  
**SET 4**

In [117]:

```
## Create a wordcloud of cluster 0
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_0)):
    for i in range(len(text_0[j])):
        t_b = t_b + text_0[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



In [118]:

```
## Create a wordcloud of cluster 1
from wordcloud import WordCloud
from matplotlib.pyplot import figure

t_b = ''
for j in range(len(text_1)):
    for i in range(len(text_1[j])):
        t_b = t_b + text_1[j][i] + ' '
#print(t_b)

word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```

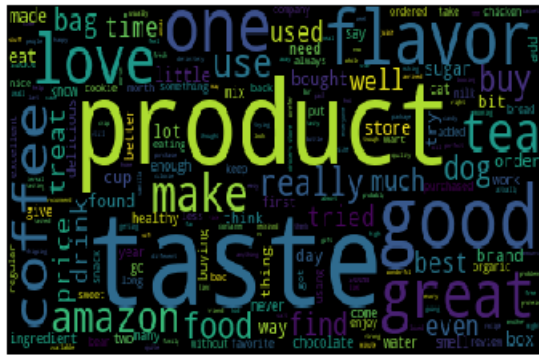


In [119]:

```

## Create a wordcloud of cluster 2
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_2)):
    for i in range(len(text_2[j])):
        t_b = t_b + text_2[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

```



In [120]:

```

## Create a wordcloud of cluster 3
from wordcloud import WordCloud
from matplotlib.pyplot import figure

t_b = ''
for j in range(len(text_3)):
    for i in range(len(text_3[j])):
        t_b = t_b + text_3[j][i] + ' '
#print(t_b)

word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

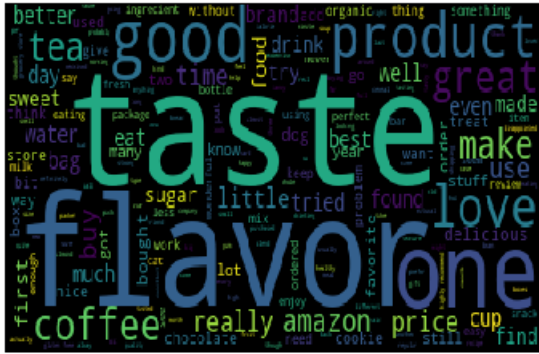
```



In [121]:

```
## Create a wordcloud of cluster 4
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_4)):
    for i in range(len(text_4[j])):
        t_b = t_b + text_4[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
```

```
plt.axis('off')
plt.show()
```



Observation: Agglomerative with TFIDF-W2V mostly contains words are of bevarages taste,falvour,good product catogory.### Apply agglomerative algorithm and try number of clusters like 2.

In [91]:

```
from sklearn.cluster import AgglomerativeClustering
agg = AgglomerativeClustering(n_clusters = 2)
p = agg.fit_predict(tf_w2v)
```

In [92]:

```
### append a same label cluster in index
index_0 = []
index_1 = []

for i in range(len(p)):
    if p[i] == 0:
        index_0.append(i)
for i in range(len(p)):
    if p[i] == 1:
        index_1.append(i)
```

In [93]:

```
text_0 = []
text_1 = []

for i in range(len(index_0)):
    text_0.append(list_of_sent[index_0[i]])
for i in range(len(index_1)):
    text_1.append(list_of_sent[index_1[i]])
```

In [94]:

```
## Create a wordcloud of cluster 0
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_0)):
    for i in range(len(text_0[j])):
        t_b = t_b + text_0[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```





In [95]:

```
## Create a wordcloud of cluster 1
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
for j in range(len(text_1)):
    for i in range(len(text_1[j])):
        t_b = t_b + text_1[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



Observation: Agglomerative with TFIDF-W2V mostly contains words are of bevarages taste,falvour,good product catogory.

### [5.3] DBSCAN Clustering

### [5.3.1] Applying DBSCAN on AVG W2V, SET 3

In [138]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
w2v = sc.fit_transform(sent_vectors)
```

In [141]:

```
## elbow method eps points
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors

eps = 100
knn = NearestNeighbors(n_neighbors = eps)
knn.fit(w2v)

dist , index = knn.kneighbors(w2v)

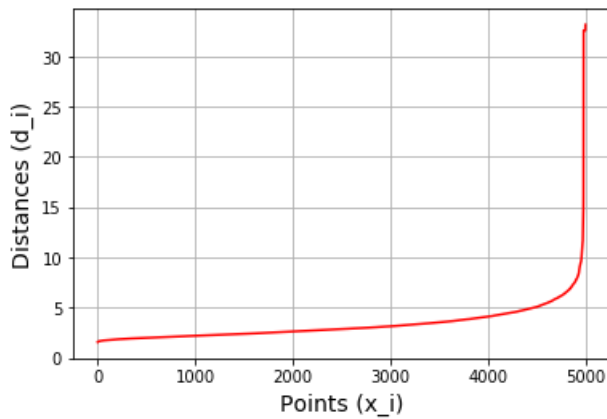
sort = sorted(dist[:,eps-1])

plt.plot(list(range(1,len(w2v)+1)),sort,'r-')
plt.xlabel('Points (x_i)',size=14)
plt.ylabel('Distances (d_i)',size=14)
plt.title('Distances VS Points Plot\n',size=18)
plt.grid()
plt.show()
```

### Distances V/C Dointe Plot



## DISTANCES VS POINTS PLOT



Observation: From above we can see that the right value of Epsilon is 7 because after that there is sharp rise in the value of distances (d\_i's)

In [142]:

```
# Function definition for implementing DBSCAN
def dbscan(epsilon, samples, Data):
    from sklearn.cluster import DBSCAN
    db = DBSCAN(eps=epsilon, min_samples=samples, n_jobs=-1).fit(sent_vectors)

    # Number of clusters in labels, ignoring noise (-1) if present.
    n_clusters = len(set(db.labels_))
    print("Number of clusters for MinPts = %d and Epsilon = %f is : %d"
          % (samples, epsilon, n_clusters))
    print("Labels(-1 is for Noise) : ", set(db.labels_))
    print()
    return db
```

In [148]:

```
db = dbscan(7, 232, w2v)
```

Number of clusters for MinPts = 232 and Epsilon = 7.000000 is : 1  
Labels(-1 is for Noise) : {0}

In [149]:

```
### append a same label cluster in index
index_0 = []

for i in range(len(p)):
    if p[i] == 0:
        index_0.append(i)
```

In [150]:

```
text_0 = []

for i in range(len(index_0)):
    text_0.append(list_of_sent[index_0[i]])
```

## [5.3.2] Wordclouds of clusters obtained after applying DBSCAN on AVG W2V SET 3

In [151]:

```
## Create a wordcloud of cluster 0
from wordcloud import WordCloud
from matplotlib.pyplot import figure
t_b = ''
```

```
for j in range(len(text_0)):
    for i in range(len(text_0[j])):
        t_b = t_b + text_0[j][i] + ' '
#print(t_b)
word_cloud = WordCloud(relative_scaling = 1.0).generate(t_b)
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



Observation: DBSCAN with AVG-W2V mostly contains words are of bevarages taste,falvour,good product catogory.

### [5.3.3] Applying DBSCAN on TFIDF W2V, SET 4

In [152]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
w2v = sc.fit_transform(tfidf_sent_vectors)
```

In [153]:

```
### elbow method eps points
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors

eps = 100

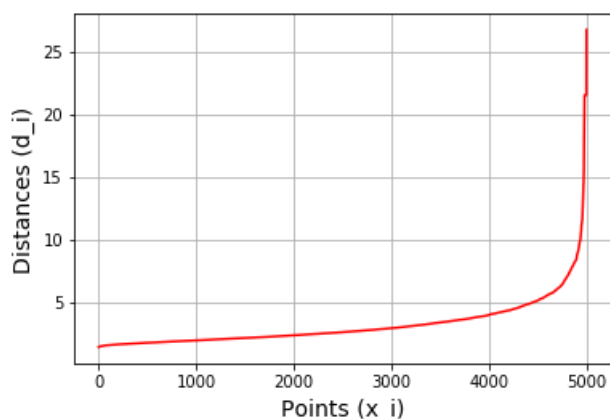
knn = NearestNeighbors(n_neighbors = eps)
knn.fit(w2v)

dist , index = knn.kneighbors(w2v)

sort = sorted(dist[:,eps-1])

plt.plot(list(range(1,len(w2v)+1)),sort,'r-')
plt.xlabel('Points (x_i)',size=14)
plt.ylabel('Distances (d_i)',size=14)
plt.title('Distances VS Points Plot\n',size=18)
plt.grid()
plt.show()
```

### Distances VS Points Plot



Observation: From above we can see that the right value of Epsilon is 7 because after that there is sharp rise in the value of distances

In [154]:

In [155]:

In [156]:

In [157]:

In [159]:

[illegible]



In [ ]:

Observation:

DBSCAN **with** TFIDF-W2V mostly contains words are of bevarages taste,falvour,good product catogory.

## [6] Conclusions

In [ ]:

```
### kmeans table
```

In [1]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer","Hyperparameter"]

x.add_row(["BOW", "9"])
x.add_row(["TFIDF", "9"])
x.add_row(["AvgW2V", "9"])
x.add_row(["TFIDF W2V", "9"])
print(x)
```

Vectorizer	Hyperparameter
BOW	9
TFIDF	9
AvgW2V	9
TFIDF W2V	9

In [ ]:

```
### Agglomerative table
```

In [97]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer","Hyperparameter","clusters"]

x.add_row(["AvgW2V", "5", "well grouped"])
x.add_row(["AvgW2V", "2", "well grouped"])
x.add_row(["TFIDF W2V", "5", "well grouped"])
x.add_row(["TFIDF W2V", "2", "well grouped"])
print(x)
```

Vectorizer	Hyperparameter	clusters
AvgW2V	5	well grouped
AvgW2V	2	well grouped
TFIDF W2V	5	well grouped
TFIDF W2V	2	well grouped

In [ ]:

```
### DBSCAN table
```

```
In [98]:
```

```
from prettytable import PrettyTable
```

```
x = PrettyTable()
```

```
x.field_names = ["Vectorizer", "eps_Hyperparameter"]
```

```
x.add_row(["AvgW2V", "7"])
```

```
x.add_row(["TFIDF W2V", "7"])
```

```
print(x)
```

```
+-----+-----+
| Vectorizer | eps_Hyperparameter |
+-----+-----+
| AvgW2V     | 7                   |
| TFIDF W2V  | 7                   |
+-----+-----+
```

1) Clusters are well grouped in agglomerative and DBSCAN. 2) In Kmeans, some clusters are contain only one word and some are well grouped.