Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (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 - unqiue 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 cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral 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 wil be set to "positive". Otherwise, it will be set to "negative".

In [0]:

```
%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]:

```
from google.colab import drive


drive.mount('/content/drive')


%cd ./drive/My Drive
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?cli
ent_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuserconten
t.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20http
s%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.google
apis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.
photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.read
only&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
/content/drive/My Drive
```

In [4]:

```
# 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 50
0000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500
0""", 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 (5000, 10)
```

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDen
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

In [0]:

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

In [6]:

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

(80668, 7)

Out[6]:

	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 [7]:

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

Out[7]:

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

In [8]:

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

Out[8]:

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 [9]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessD
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	

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 delelte 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 [0]:

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

In [11]:

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

Out[11]:

```
(4986, 10)
```

In [12]:

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

Out[12]:

```
99.72
```

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 calcualtions

In [13]:

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

display.head()
```

Out[13]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessD
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [0]:

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

In [15]:

```
#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()
```

```
(4986, 10)
```

Out[15]:

```
1    4178
0     808
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 obsereved to be better than Porter Stemming)

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

In [16]:

```
# 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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br
/>http://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<br /><
br />The Victor M380 and M502 traps are unreal, of course -- total fly gen
ocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious thes
e chips are.  The best thing was that there were a lot of "brown" chips in
the bsg (my favorite), so I bought some more through amazon and shared wit
h family and friends.  I am a little disappointed that there are not, so f
ar, very many brown chips in these bags, but the flavor is still very goo
d.  I like them better than the yogurt and green onion flavor because they
do not seem to be as salty, and the onion flavor is better.  If you have
n't eaten Kettle chips before, I recommend that you try a bag before buyin
g bulk.  They are thicker and crunchier than Lays but just as fresh out of
the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they w
ere ordering; the other wants crispy cookies.  Hey, I'm sorry; but these r
eviews do nobody any good beyond reminding us to look  before ordering.<br
/><br />These are chocolate-oatmeal cookies.  If you don't like that combi
nation, don't order this type of cookie.  I find the combo quite nice, rea
lly.  The oatmeal sort of "calms" the rich chocolate flavor and gives the
cookie sort of a coconut-type consistency.  Now let's also remember that t
astes differ; so, I've given my opinion.<br /><br />Then, these are soft,
chewy cookies -- as advertised.  They are not "crispy" cookies, or the blu
rb would say "crispy," rather than "chewy."  I happen to like raw cookie d
ough; however, I don't see where these taste like raw cookie dough.  Both
are soft, however, so is this the confusion?  And, yes, they stick togethe
r.  Soft cookies tend to do that.  They aren't individually wrapped, which
would add to the cost.  Oh yeah, chocolate chip cookies tend to be somewha
t sweet.<br /><br />So, if you want something hard and crisp, I suggest Na
biso's Ginger Snaps.  If you want a cookie that's soft, chewy and tastes l
ike a combination of chocolate and oatmeal, give these a try.  I'm here to
place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k
cup is great coffee.  dcaf is very good as well
==================================================
```

In [17]:

```
# 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_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br
/> /><br />The Victor M380 and M502 traps are unreal, of course -- total f
ly genocide. Pretty stinky, but only right nearby.
```

In [18]:

```
# 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)
```

```
Why is this $[...] when the same product is available for $[...] here? />T
he Victor M380 and M502 traps are unreal, of course -- total fly genocide.
Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious thes
e chips are.  The best thing was that there were a lot of "brown" chips in
the bsg (my favorite), so I bought some more through amazon and shared wit
h family and friends.  I am a little disappointed that there are not, so f
ar, very many brown chips in these bags, but the flavor is still very goo
d.  I like them better than the yogurt and green onion flavor because they
do not seem to be as salty, and the onion flavor is better.  If you have
n't eaten Kettle chips before, I recommend that you try a bag before buyin
g bulk.  They are thicker and crunchier than Lays but just as fresh out of
the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they w
ere ordering; the other wants crispy cookies.  Hey, I'm sorry; but these r
eviews do nobody any good beyond reminding us to look  before ordering.The
se are chocolate-oatmeal cookies.  If you don't like that combination, do
n't order this type of cookie.  I find the combo quite nice, really.  The
oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sor
t of a coconut-type consistency.  Now let's also remember that tastes diff
er; so, I've given my opinion.Then, these are soft, chewy cookies -- as ad
vertised.  They are not "crispy" cookies, or the blurb would say "crispy,"
rather than "chewy."  I happen to like raw cookie dough; however, I don't
see where these taste like raw cookie dough.  Both are soft, however, so i
s this the confusion?  And, yes, they stick together.  Soft cookies tend t
o do that.  They aren't individually wrapped, which would add to the cost.
Oh yeah, chocolate chip cookies tend to be somewhat sweet.So, if you want
something hard and crisp, I suggest Nabiso's Ginger Snaps.  If you want a
cookie that's soft, chewy and tastes like a combination of chocolate and o
atmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cup i
s great coffee.  dcaf is very good as well
```

In [0]:

```
# 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 [20]:

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

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they w
ere ordering; the other wants crispy cookies.  Hey, I am sorry; but these
reviews do nobody any good beyond reminding us to look  before ordering.<b
r /><br />These are chocolate-oatmeal cookies.  If you do not like that co
mbination, do not order this type of cookie.  I find the combo quite nice,
really.  The oatmeal sort of "calms" the rich chocolate flavor and gives t
he cookie sort of a coconut-type consistency.  Now let is also remember th
at tastes differ; so, I have given my opinion.<br /><br />Then, these are
soft, chewy cookies -- as advertised.  They are not "crispy" cookies, or t
he blurb would say "crispy," rather than "chewy."  I happen to like raw co
okie dough; however, I do not see where these taste like raw cookie dough.
Both are soft, however, so is this the confusion?  And, yes, they stick to
gether.  Soft cookies tend to do that.  They are not individually wrapped,
which would add to the cost.  Oh yeah, chocolate chip cookies tend to be s
omewhat sweet.<br /><br />So, if you want something hard and crisp, I sugg
est Nabiso is Ginger Snaps.  If you want a cookie that is soft, chewy and
tastes like a combination of chocolate and oatmeal, give these a try.  I a
m here to place my second order.
==================================================
```

In [21]:

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

```
Why is this $[...] when the same product is available for $[...] here?<br
/> /><br />The Victor  and  traps are unreal, of course -- total fly genoc
ide. Pretty stinky, but only right nearby.
```

In [22]:

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

```
Wow So far two two star reviews One obviously had no idea what they were o
rdering the other wants crispy cookies Hey I am sorry but these reviews do
nobody any good beyond reminding us to look before ordering br br These ar
e chocolate oatmeal cookies If you do not like that combination do not ord
er this type of cookie I find the combo quite nice really The oatmeal sort
of calms the rich chocolate flavor and gives the cookie sort of a coconut
type consistency Now let is also remember that tastes differ so I have giv
en my opinion br br Then these are soft chewy cookies as advertised They a
re not crispy cookies or the blurb would say crispy rather than chewy I ha
ppen to like raw cookie dough however I do not see where these taste like
raw cookie dough Both are soft however so is this the confusion And yes th
ey stick together Soft cookies tend to do that They are not individually w
rapped which would add to the cost Oh yeah chocolate chip cookies tend to
be somewhat sweet br br So if you want something hard and crisp I suggest
Nabiso is Ginger Snaps If you want a cookie that is soft chewy and tastes
like a combination of chocolate and oatmeal give these a try I am here to
place my second order
```

In [0]:

```
# 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 revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
s', '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', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', '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", 'no
w', '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', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [24]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    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 stopwor
ds)
    preprocessed_reviews.append(sentance.strip())
```

100%|██████████| 4986/4986 [00:01<00:00, 2506.54it/s]

In [25]:

```
preprocessed_reviews[1500]
```

Out[25]:

```
'wow far two two star reviews one obviously no idea ordering wants crispy
cookies hey sorry reviews nobody good beyond reminding us look ordering ch
ocolate oatmeal cookies not like combination not order type cookie find co
mbo quite nice really oatmeal sort calms rich chocolate flavor gives cooki
e sort coconut type consistency let also remember tastes differ given opin
ion soft chewy cookies advertised not crispy cookies blurb would say crisp
y rather chewy happen like raw cookie dough however not see taste like raw
cookie dough soft however confusion yes stick together soft cookies tend n
ot individually wrapped would add cost oh yeah chocolate chip cookies tend
somewhat sweet want something hard crisp suggest nabiso ginger snaps want
cookie soft chewy tastes like combination chocolate oatmeal give try place
second order'
```

[3.2] Preprocessing Review Summary

In [0]:

```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [27]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

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

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbot
t', 'abby', 'abdominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [28]:

```
#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/modul
es/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(preprocessed_reviews)
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_c
ounts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [29]:

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

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.g
et_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able
find', 'able get', 'absolute', 'absolutely', 'absolutely delicious', 'abso
lutely love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [0]:

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

In [31]:

```
# 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 varible 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_sentance,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.bi
n', binary=True)
        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 = True, to
train your own w2v ")
```

```
[('excellent', 0.9939777851104736), ('amazing', 0.9939639568328857), ('ove
rall', 0.9936991333961487), ('wonderful', 0.9936230778694153), ('looking',
0.993614137172699), ('think', 0.9936040639877319), ('though', 0.9935807585
716248), ('calorie', 0.9932543635368347), ('definately', 0.993165850639343
3), ('fantastic', 0.9930830597877502)]
==================================================
[('experience', 0.9993714094161987), ('varieties', 0.9993330836296082),
('sometimes', 0.9992867708206177), ('second', 0.9992606043815613), ('awfu
l', 0.9992459416389465), ('part', 0.9992445111274719), ('yes', 0.999228775
5012512), ('sticks', 0.9992273449897766), ('inside', 0.9992148876190186),
('wife', 0.9991919994354248)]
```

In [32]:

```
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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stink
y', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received',
'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead',
'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'win
dows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywher
e', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstand
ing', 'window', 'everybody', 'asks', 'bought', 'made']
```

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

[4.4.1.1] Avg W2v

[4.4.1.2] TFIDF weighted W2v

[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. (https://stackoverflow.com/questions/12893492/choosing-eps-and-minpts-for-dbscan-r/48558030#48558030)
   - 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 [0]:

```
#Loading the data into X,Y
X = preprocessed_reviews
Y = final['Score'].values
```

In [0]:

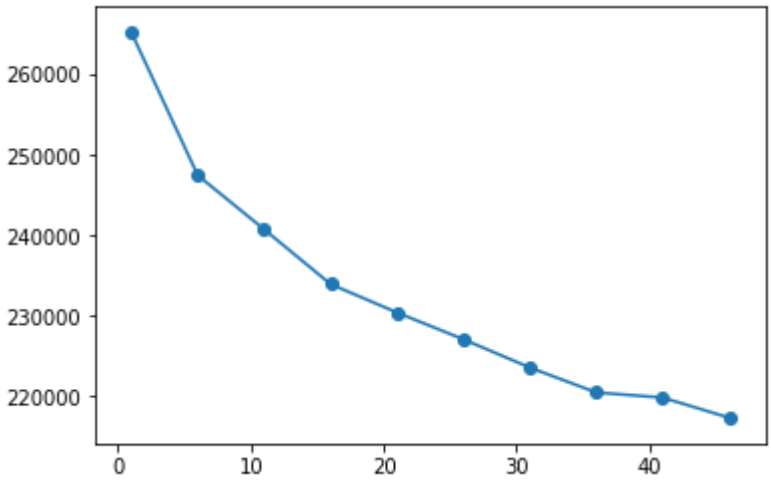
```
from sklearn.model_selection import learning_curve
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
# we use the fitted CountVectorizer to convert the text to vector
X_bow = vectorizer.fit_transform(X)
```

In [35]:

```
from sklearn.cluster import KMeans
#Hyper parameter tuning
inertia = []

K = range(1,50,5)
for i in K:

  kmeans = KMeans(n_clusters=i, random_state=0).fit(X_bow)
  inertia.append(kmeans.inertia_)

#plot of hyperparameter vs square of intra cluster distances
plt.plot(K, inertia)
plt.scatter(K,inertia)
```

Out[35]:

```
<matplotlib.collections.PathCollection at 0x7facbb4d6550>
```



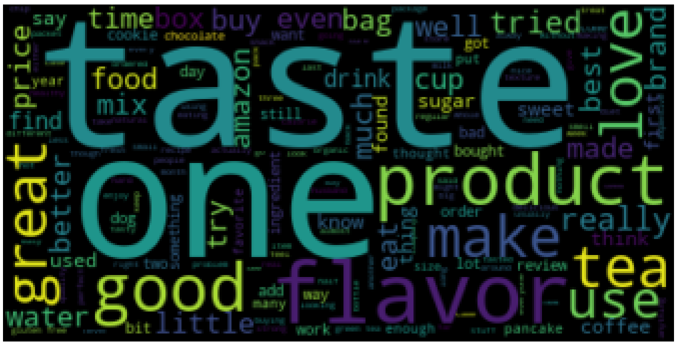
In [0]:

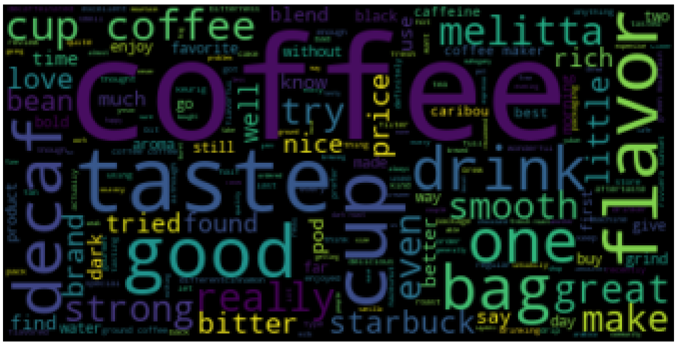
```
n_clusters=5
```

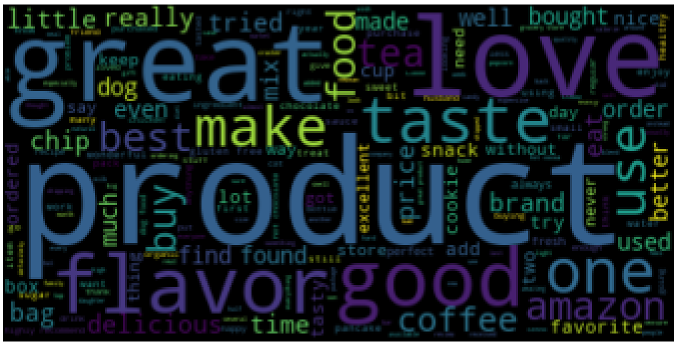
[5.1.2] Wordclouds of clusters obtained after applying k-means on BOW SET 1

In [37]:

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

kmeans = KMeans(n_clusters, random_state=0).fit(X_bow)

#constructing a pandas dataframe containing Reviews and cluster labels
df=pd.DataFrame(X,columns=['Reviews'])
df['cluster']=kmeans.labels_

#https://www.datacamp.com/community/tutorials/wordcloud-python

#joining all the reviews by cluster name in the form of text
for i in range(0,n_clusters):
  a = " ".join( df[df["cluster"]==i].Reviews)

  # Create and generate a word cloud image:
  wordcloud = WordCloud().generate(a)

  # Display the generated image:
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```


taste
one
product
flavor
good
great
love
tea
use
make
really
food
mix
drink
cup
sugar
best
tried
time
buy
bag
well
water
little
price
find
made
sweet
coffee


coffee
taste
good
flavor
decaf
drink
cup
one
bag
great
make
smooth
strong
really
bitter
starbuck
melitta
rich
try
well
price
nice
love
bean
time
brand
tried
found
little


product
great
love
flavor
good
taste
make
one
amazon
coffee
use
best
food
tea
buy
better
brand
little
really
tried
made
well
bought
chip
dog
delicious
time
order


chip
bag
flavor
potato chip
kettle
kettle brand
kettle chip
taste
great
good
snack
calorie
salty
love
amazon
product
really
one
try
buy
salt
vinegar
better
favorite
much
time
best
sea salt

In the above clusters, cluster wise

1.taste, one, product, good, flavour, make, tea, use, great, love are the words seen which implies that tea flavour is good.

2.coffee,taste,good,drink,love,one,bag implies that coffee is loved.

3.great,product,love,make,taste,good,coffee implies it is a good review about coffee.

4.great product,love,taste,flavour,good,cooffe implies it is a positive review cluster about coffee.

5.bag,chip,flavor,great,delicious implies the positive review are there in this cluster.

6.food,one,product,goodbuy,flavor,organic,dog ,amazon perhaps implies about food or dog food.

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

In [0]:

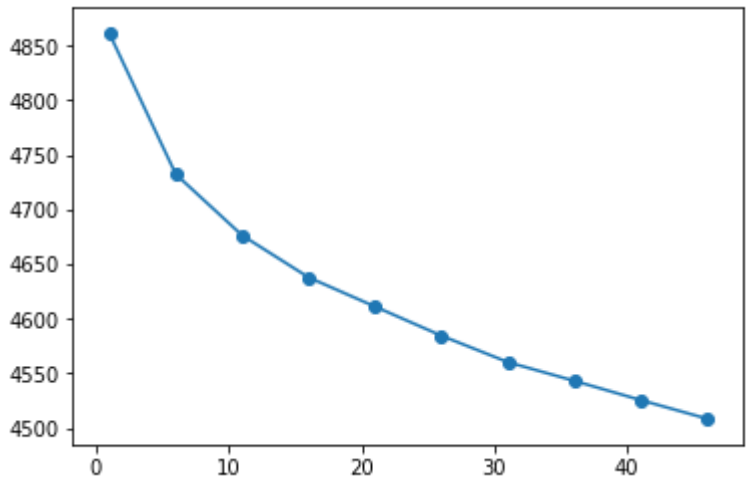
```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer


#applying tfidf vectorizer upon the preprocessed reviews
tf_idf_vect = TfidfVectorizer()

X_tf = tf_idf_vect.fit_transform(X)
```

In [39]:

```
from sklearn.cluster import KMeans
#Hyper parameter tuning
inertia = []

K = range(1,50,5)
for i in K:

  kmeans = KMeans(n_clusters=i, random_state=0,).fit(X_tf)
  inertia.append(kmeans.inertia_)

  #plotting a plot of no of clusters vs inertia
plt.plot(K, inertia)
plt.scatter(K,inertia)
```

Out[39]:

```
<matplotlib.collections.PathCollection at 0x7facbd8af4e0>
```



In [0]:

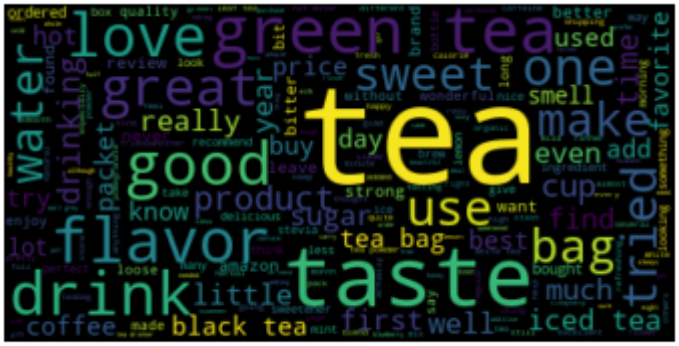
```
n_clusters=5
```

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

In [41]:

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

kmeans = KMeans(n_clusters, random_state=0).fit(X_tf)

#constructing a pandas dataframe containing Reviews and cluster labels

df=pd.DataFrame(X,columns=['Reviews'])
df['cluster']=kmeans.labels_
print(df['cluster'].value_counts())

#https://www.datacamp.com/community/tutorials/wordcloud-python

#joining all the reviews by cluster name in the form of text
for i in range(0,n_clusters):
  a = " ".join( df[df["cluster"]==i].Reviews)

  # Create and generate a word cloud image:
  wordcloud = WordCloud().generate(a)

  # Display the generated image:
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```

```
4    3563
0     457
3     353
2     344
1     269
Name: cluster, dtype: int64
```



In the above clusters, cluster wise

1.bag,chip,flavor,great,delicious,potato implies the positive review for potato chips are there in this cluster.

2.flavor,tea,good,drink,love,one,taste implies that coffee is loved.

3.decaf ,flavor,taste,good,coffee implies it is a good review about coffee.

4.food,dog,product,dog food perhaps implies about dog food.

5.cookie,product,good,taste implies about good taste of cookie.

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

In [42]:

```
#Loading the data into X,Y
X = preprocessed_reviews
Y = final['Score'].values


i=0
list_of_sentance=[]
for sentance in X:
    list_of_sentance.append(sentance.split())

# Training W2V model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)

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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stink
y', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received',
'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead',
'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'win
dows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywher
e', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstand
ing', 'window', 'everybody', 'asks', 'bought', 'made']
```

In [43]:

```
from tqdm import tqdm
import numpy as np

# converting train data text

# 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_sentance): # 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)
sent_vectors= np.array(sent_vectors)
print(sent_vectors.shape)
print(sent_vectors[0])
```

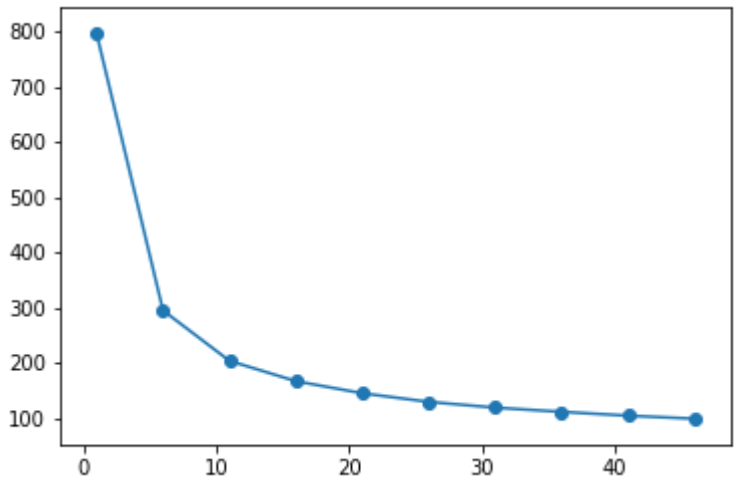
```
100%|██████████| 4986/4986 [00:04<00:00, 1245.85it/s]

(4986, 50)
[-0.24144135 -0.15184975 -0.53735699  0.10232854  0.02866363 -0.20308954
  0.08273677  0.14593063  0.3816066  -0.1729753   0.20630757 -0.66736398
 -0.06261844  0.20696688  0.22899507 -0.22909593  0.19172875 -0.35329887
 -0.51258685  0.48675316  0.22619376 -0.35533181 -0.01658106 -0.47987811
 -0.58082726 -0.56143429 -0.44193372 -0.16686884 -0.49500509 -0.45521416
 -0.17945617 -0.45480419 -0.51866921 -0.0297471   0.29545126  0.50868614
 -0.24032706 -0.70775735  0.12604875 -0.13386704 -0.23866293 -0.18100964
 -0.32390143  0.26992674  0.09792589 -0.25528284 -0.08630418  0.02616523
  0.2261916   0.26160478]
```

In [44]:

```
from sklearn.cluster import KMeans
#Hyper parameter tuning
inertia = []

K = range(1,50,5)
for i in K:

  kmeans = KMeans(n_clusters=i, random_state=0,).fit(sent_vectors)
  inertia.append(kmeans.inertia_)

  #plotting no of clusters vssquare of intra clsuter distances
plt.plot(K, inertia)
plt.scatter(K,inertia)
```

Out[44]:

```
<matplotlib.collections.PathCollection at 0x7facbbe7ec50>
```



In [0]:

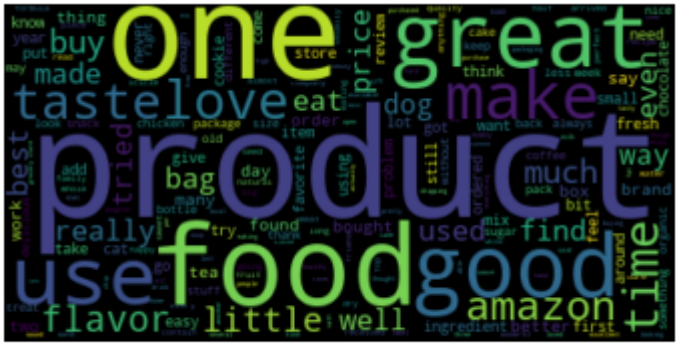
```
n_clusters=6
```

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

In [47]:

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

kmeans = KMeans(n_clusters, random_state=0).fit(sent_vectors)

#constructing a pandas dataframe containing Reviews and cluster labels

df=pd.DataFrame(X,columns=['Reviews'])
df['cluster']=kmeans.labels_
print(df['cluster'].value_counts())

#https://www.datacamp.com/community/tutorials/wordcloud-python


#joining all the reviews by cluster name in the form of text
for i in [0,1,2,3,4]:
  a = " ".join( df[df["cluster"]==i].Reviews)

  # Create and generate a word cloud image:
  wordcloud = WordCloud().generate(a)

  # Display the generated image:
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```

```
2    1528
4    1190
1    1083
0     921
3     249
5      15
Name: cluster, dtype: int64
```



1.good,taste,coffee,flavor,chip implies good coffe and chip reviews.

2.one,great,product,use,good,food implies food is great.

3.flavor,taste,coffee,tea,great implies good coffee taste.

4.great,gluten,free,pancake implies the positive review for gluten free pancake.

5.amazon,great,dog,food,love,product implies good dog food product.

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

In [48]:

```
#preparing reviews for genism model
X = preprocessed_reviews
Y = final['Score'].values
i=0
list_of_sentance_tf=[]
for sentance in X:
    list_of_sentance_tf.append(sentance.split())


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

# this line of code trains w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_tf,min_count=5,size=50, workers=4)

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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stink
y', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received',
'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead',
'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'win
dows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywher
e', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstand
ing', 'window', 'everybody', 'asks', 'bought', 'made']
```

In [0]:

```
from tqdm import tqdm
import numpy as np
model = TfidfVectorizer()
model.fit(X)
tf_idf_matrix = model.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 [50]:

```
# 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= []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_tf): # 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.append(sent_vec)
    row += 1
```

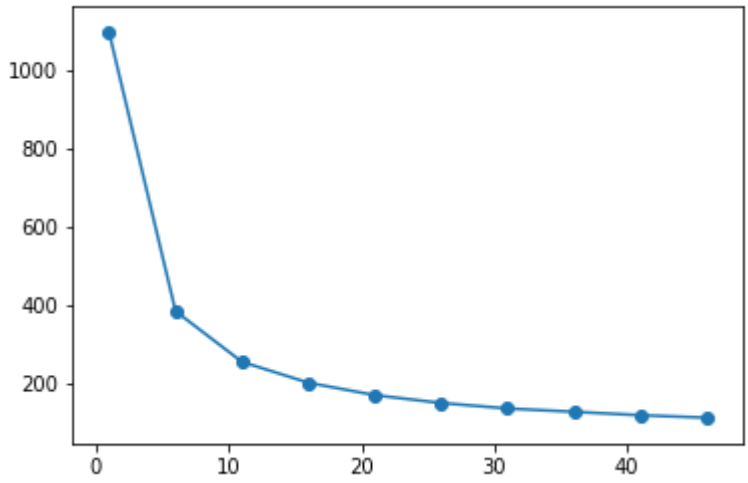
```
100%|██████████| 4986/4986 [00:26<00:00, 185.73it/s]
```

In [51]:

```
from sklearn.cluster import KMeans
#Hyper parameter tuning
inertia = []

K = range(1,50,5)
for i in K:

  kmeans = KMeans(n_clusters=i, random_state=0,).fit(tfidf_sent)
  inertia.append(kmeans.inertia_)

plt.plot(K, inertia)
plt.scatter(K,inertia)
```

Out[51]:

```
<matplotlib.collections.PathCollection at 0x7facb42d36d8>
```



In [0]:

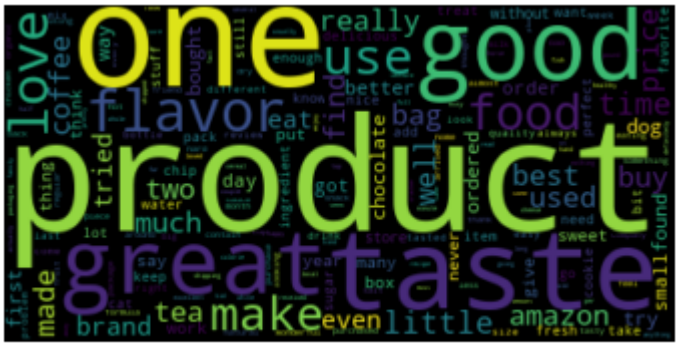
```
n_clusters=5
```

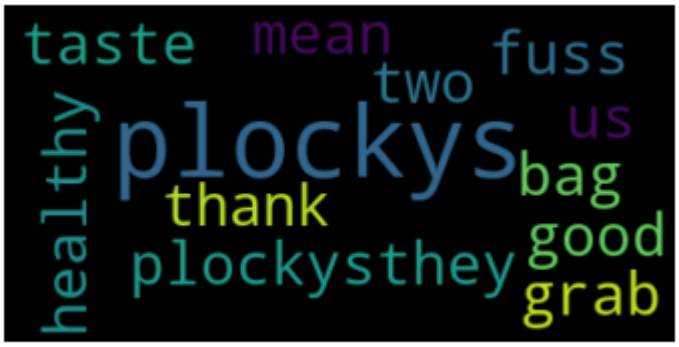
[5.1.8] Wordclouds of clusters obtained after applying k-means on TFIDF W2V SET 4

In [53]:

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

kmeans = KMeans(n_clusters, random_state=0).fit(tfidf_sent)

#constructing a pandas dataframe containing Reviews and cluster labels

df=pd.DataFrame(X,columns=['Reviews'])
df['cluster']=kmeans.labels_
print(df['cluster'].value_counts())

#https://www.datacamp.com/community/tutorials/wordcloud-python
#joining all the reviews by cluster name in the form of text
for i in range(0,n_clusters):
  a = " ".join( df[df["cluster"]==i].Reviews)

  # Create and generate a word cloud image:
  wordcloud = WordCloud().generate(a)

  # Display the generated image:
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```

```
0    1937
1    1120
3     994
2     919
4      16
Name: cluster, dtype: int64
```





1.This cluster implies coffee flavor is great.

2.great flavor,coffee,chip

3.love dog product food amazon

4.one,good,product,great,flavor,taste

[5.2] Agglomerative Clustering

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

In [54]:

```
#Loading the data into X,Y
X = preprocessed_reviews



i=0
list_of_sentance=[]
for sentance in X:
    list_of_sentance.append(sentance.split())

# Training W2V model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)

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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stink
y', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received',
'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead',
'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'win
dows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywher
e', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstand
ing', 'window', 'everybody', 'asks', 'bought', 'made']
```

In [55]:

```
from tqdm import tqdm
import numpy as np

# converting train data text

# 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_sentance): # 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)
sent_vectors= np.array(sent_vectors)
print(sent_vectors.shape)
print(sent_vectors[0])
```

```
100%|██████████| 4986/4986 [00:03<00:00, 1298.69it/s]

(4986, 50)
[-0.25830849 -0.16807394 -0.57479847  0.10234873  0.03658586 -0.18977748
  0.10063386  0.1211867   0.36341278 -0.14940986  0.2270433  -0.65956175
 -0.1075978   0.20495849  0.23367976 -0.22859613  0.21094541 -0.36575915
 -0.51385469  0.53105988  0.23626625 -0.37312785 -0.03676397 -0.51306904
 -0.57592615 -0.56516721 -0.46115332 -0.13417143 -0.47171711 -0.42050802
 -0.15126521 -0.45948119 -0.4785623  -0.01218868  0.27680413  0.49889576
 -0.22501354 -0.69292752  0.09246605 -0.09526179 -0.26660516 -0.22810629
 -0.31486621  0.27059323  0.09521337 -0.27315956 -0.09658476  0.05536896
  0.24189898  0.25251905]
```

In [0]:

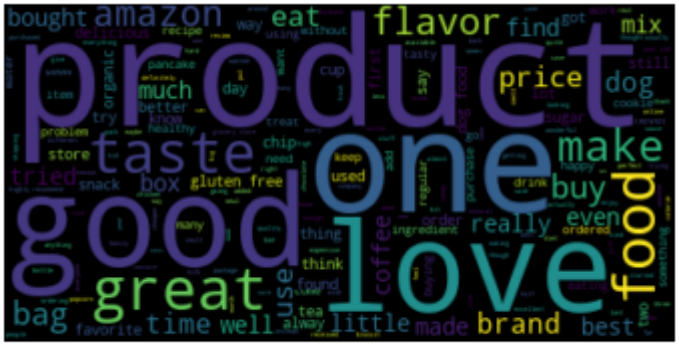
```
from sklearn.cluster import AgglomerativeClustering
#Hyper parameter tuning



agg = AgglomerativeClustering(n_clusters=5).fit(sent_vectors)
```

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

In [57]:

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

agg = AgglomerativeClustering(n_clusters=3).fit(sent_vectors)

df=pd.DataFrame(X,columns=['Reviews'])
df['cluster']=agg.labels_
print(df['cluster'].value_counts())

#https://www.datacamp.com/community/tutorials/wordcloud-python

for i in range (0,3):
  a = " ".join( df[df["cluster"]==i].Reviews)

  # Create and generate a word cloud image:
  wordcloud = WordCloud().generate(a)

  # Display the generated image:
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```

```
1    2672
0    1239
2    1075
Name: cluster, dtype: int64
```



1.chocolate,flavor,one,taste,product,good.

2.great,good,product,eat,flavor

3.good,chip,taste,coffee,flavor implies good set of reviews for coffee.

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

In [58]:

```
#preparing reviews for genism model
X = preprocessed_reviews
Y = final['Score'].values
i=0
list_of_sentance_tf=[]
for sentance in X:
    list_of_sentance_tf.append(sentance.split())


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

# this line of code trains w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_tf,min_count=5,size=50, workers=4)

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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stink
y', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received',
'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead',
'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'win
dows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywher
e', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstand
ing', 'window', 'everybody', 'asks', 'bought', 'made']
```

In [0]:

```
from tqdm import tqdm
import numpy as np
model = TfidfVectorizer()
model.fit(X)
tf_idf_matrix = model.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 [60]:

```
# 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= []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_tf): # 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.append(sent_vec)
    row += 1
```

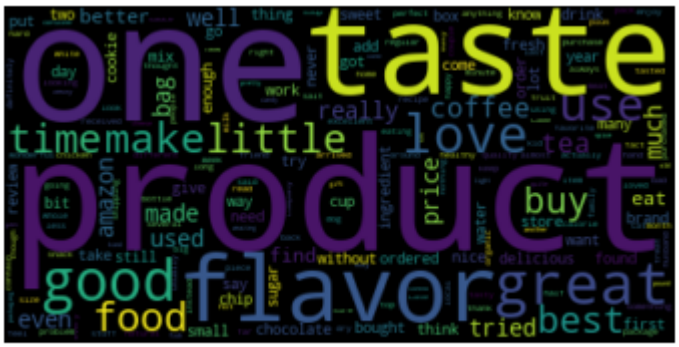
```
100%|██████████| 4986/4986 [00:25<00:00, 196.94it/s]
```

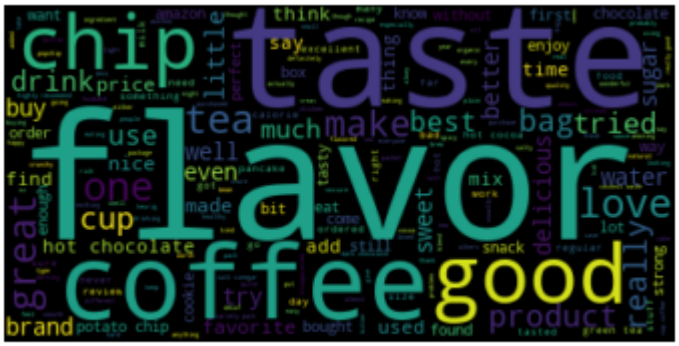
[5.2.4] Wordclouds of clusters obtained after applying Agglomerative Clustering on TFIDF W2V SET 4

In [61]:

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

agg = AgglomerativeClustering(n_clusters=4).fit(tfidf_sent)

df=pd.DataFrame(X,columns=['Reviews'])
df['cluster']=agg.labels_
print(df['cluster'].value_counts())

#https://www.datacamp.com/community/tutorials/wordcloud-python

for i in range (0,4):
  a = " ".join( df[df["cluster"]==i].Reviews)

  # Create and generate a word cloud image:
  wordcloud = WordCloud().generate(a)

  # Display the generated image:
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```

```
2    1918
1    1525
0    1378
3     165
Name: cluster, dtype: int64
```





[5.3] DBSCAN Clustering

[5.3.1] Applying DBSCAN on AVG W2V, SET 3

In [62]:

```
#Loading the data into X,Y
X = preprocessed_reviews
Y = final['Score'].values


i=0
list_of_sentance=[]
for sentance in X:
    list_of_sentance.append(sentance.split())

# Training W2V model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)

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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stink
y', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received',
'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead',
'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'win
dows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywher
e', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstand
ing', 'window', 'everybody', 'asks', 'bought', 'made']
```

In [63]:

```
from tqdm import tqdm
import numpy as np

# converting train data text

# 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_sentance): # 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)
sent_vectors= np.array(sent_vectors)
print(sent_vectors.shape)
print(sent_vectors[0])
```

```
100%|██████████| 4986/4986 [00:03<00:00, 1256.46it/s]

(4986, 50)
[-0.24235901 -0.19508504 -0.59854366  0.08302972  0.03074521 -0.17533767
  0.12231856  0.11593842  0.40558213 -0.17634991  0.20485843 -0.66203579
 -0.09087931  0.21792519  0.2385435  -0.20142156  0.1764378  -0.38343457
 -0.50834025  0.509664    0.22477547 -0.36450069 -0.02603466 -0.46742483
 -0.60655289 -0.57017053 -0.46260121 -0.12841601 -0.46079649 -0.4487682
 -0.17020793 -0.4777455  -0.50828703 -0.04899138  0.28961836  0.4563262
 -0.23222275 -0.69659654  0.07912242 -0.12927693 -0.24303367 -0.18776237
 -0.3151722   0.28694269  0.08154705 -0.21242689 -0.09953801  0.03700697
  0.24171284  0.29189068]
```

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

In [0]:

```
#min_samples=2*d
min_samples=100
```

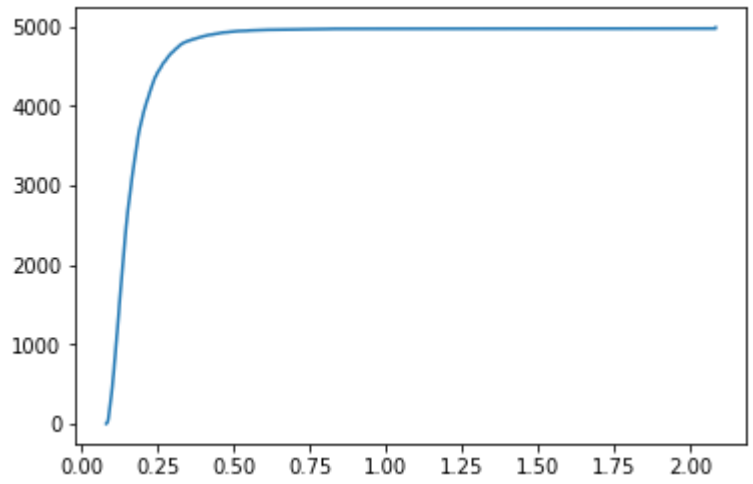
In [81]:

```
from sklearn.neighbors import NearestNeighbors
#applying nearest  neighbours to find distance from a point to 100th nearest neighbour
nn = NearestNeighbors(n_neighbors=100, algorithm='kd_tree').fit(sent_vectors)
distances, indices = nn.kneighbors(sent_vectors)

#sorting the distances
a=distances[:,99]
print(a.shape)
a.sort()

#plotting the distances
k=range(0,4986)
b=[i for i in k]
#a.tolist()
plt.plot(a,b)
#plt.scatter(b,a)
```

```
(4986,)
```

Out[81]:

```
[<matplotlib.lines.Line2D at 0x7facb4d3b630>]
```



In [73]:

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

from sklearn.cluster import DBSCAN

dbs = DBSCAN(eps=0.3,min_samples=100).fit(sent_vectors)

df=pd.DataFrame(X,columns=['Reviews'])
df['cluster']=dbs.labels_
print(df['cluster'].value_counts())
```

```
 0    4932
-1      54
Name: cluster, dtype: int64
```

In [74]:

```
#https://www.datacamp.com/community/tutorials/wordcloud-python

for i in range (0,1) :
  a = " ".join( df[df["cluster"]==i].Reviews)

  # Create and generate a word cloud image:
  wordcloud = WordCloud().generate(a)

  # Display the generated image:
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```

1.coffee,good,taste,flavor,love,one delicious implies good reviews for coffee

[5.3.3] Applying DBSCAN on TFIDF W2V, SET 4

In [75]:

```
#preparing reviews for genism model
X = preprocessed_reviews

i=0
list_of_sentance_tf=[]
for sentance in X:
    list_of_sentance_tf.append(sentance.split())


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

# this line of code trains w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_tf,min_count=5,size=50, workers=4)

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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stink
y', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received',
'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead',
'removed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'win
dows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywher
e', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstand
ing', 'window', 'everybody', 'asks', 'bought', 'made']
```

In [0]:

```
from tqdm import tqdm
import numpy as np
model = TfidfVectorizer()
model.fit(X)
tf_idf_matrix = model.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 [77]:

```
# 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= []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_tf): # 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.append(sent_vec)
    row += 1
```

100%|██████████| 4986/4986 [00:26<00:00, 186.11it/s]

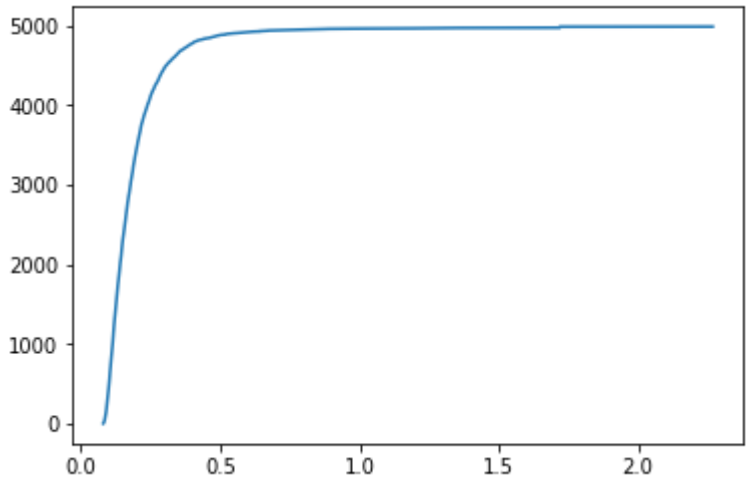
In [85]:

```
from sklearn.neighbors import NearestNeighbors
nn = NearestNeighbors(n_neighbors=100, algorithm='kd_tree').fit(tfidf_sent)
distances, indices = nn.kneighbors(tfidf_sent)

a=distances[:,99]
print(a.shape)
a.sort()

k=range(0,4986)
b=[i for i in k]
#a.tolist()
plt.plot(a,b)
#plt.scatter(b,a)
```

```
(4986,)
```

Out[85]:

```
[<matplotlib.lines.Line2D at 0x7facb46b17b8>]
```



In [86]:

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

from sklearn.cluster import DBSCAN

dbs = DBSCAN(eps=0.4,min_samples=100).fit(tfidf_sent)

df=pd.DataFrame(X,columns=['Reviews'])
df['cluster']=dbs.labels_
print(df['cluster'].value_counts())
```

```
 0    4936
-1      50
Name: cluster, dtype: int64
```

[5.3.4] Wordclouds of clusters obtained after applying DBSCAN on TFIDF W2V SET 4

In [87]:

```
#https://www.datacamp.com/community/tutorials/wordcloud-python

for i in range (0,1) :
  a = " ".join( df[df["cluster"]==i].Reviews)

  # Create and generate a word cloud image:
  wordcloud = WordCloud().generate(a)

  # Display the generated image:
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```

1.taste,flavor,good,chip,coffee,best,tea are the key words

[6] Conclusions

In [84]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "no_of_clusters"]

x.add_row(["BOW", "K-means", 5 ])
x.add_row(["TFIDF", "K-means", 5 ])
x.add_row(["AVGW2V", "K-means", 6 ])
x.add_row(["TFIDFW2V", "K-means", 5 ])
x.add_row(["AVGW2V", "Agglomerative", 3 ])
x.add_row(["TFIDFW2V", "Agglomerative", 4 ])
x.add_row(["AVGW2V", "DB-Scan", 2 ])
x.add_row(["TFIDFW2V", "DB-Scan", 2 ])



print(x)
```

```
+------------+---------------+----------------+
| Vectorizer |     Model     | no_of_clusters |
+------------+---------------+----------------+
|    BOW     |    K-means    |       5        |
|   TFIDF    |    K-means    |       5        |
|   AVGW2V   |    K-means    |       6        |
|  TFIDFW2V  |    K-means    |       5        |
|   AVGW2V   | Agglomerative |       3        |
|  TFIDFW2V  | Agglomerative |       4        |
|   AVGW2V   |    DB-Scan    |       2        |
|  TFIDFW2V  |    DB-Scan    |       2        |
+------------+---------------+----------------+
```