In [1]:
```
# Importing libraries
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
%matplotlib inline
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 nltk.stem.porter import PorterStemmer

import 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
```

```
C:\Users\Sai charan\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarnin
g: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:
```
con1 = sqlite3.connect('final.sqlite')

# Eliminating neutral reviews i.e. those reviews with Score = 3
filtered_data = pd.read_sql_query(" SELECT * FROM Reviews  ", con1)

print(filtered_data.shape)
filtered_data.head()
```

```
(364171, 12)
```

Out[2]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominat
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
1	138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
2	138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
3	138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg "(Kate)"	1	
4	138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

In [3]:
```
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=
False, kind='quicksort', na_position='last')

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

#Checking to see how much % of data still remains
((final.shape[0]*1.0)/(filtered_data.shape[0]*1.0)*100)
```

```
(364171, 12)
```

Out[3]: 100.0

In [4]:
```
final = final[final.HelpfulnessNumerator <= final.HelpfulnessDenominator]
```

In [5]:
```
final = final[final['ProductId'] != '2841233731']
final = final[final['ProductId'] != '0006641040']
final.shape
```

Out[5]:
```
(364136, 12)
```

Text Preprocessing: Stemming, stop-word removal and Lemmatization

In [6]:
```
from nltk.corpus import stopwords
stop = set(stopwords.words('english'))
words_to_keep = set(('not'))
stop -= words_to_keep
#initialising the snowball stemmer
sno = nltk.stem.SnowballStemmer('english')

 #function to clean the word of any html-tags
def cleanhtml(sentence):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext

#function to clean the word of any punctuation or special characters
def cleanpunc(sentence):
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [7]:
```
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTMl tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words used to des
cribe positive reviews
                    if(final['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to des
cribe negative reviews reviews
                else:
                    continue
            else:
                continue

    str1 = b" ".join(filtered_sentence) #final string of cleaned words


    final_string.append(str1)
    i+=1
```

In [8]:

```
final['CleanedText']=final_string
final['CleanedText']=final['CleanedText'].str.decode("utf-8")
#below the processed review can be seen in the CleanedText Column
print('Shape of final',final.shape)
final.head()
```

```
Shape of final (364136, 12)
```

Out[8]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomin
34	476617	515426	141278509X	AB1A5EGHHVA9M	CHelmic	1	
36	22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	
35	22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	
142	157910	171225	7310172001	A314APAWYQFKBJ	Diana Hersholt "dog lover"	1	
143	157909	171224	7310172001	AK0CENM3LUM28	Ana Mardoll	1	

In [9]:

```
# We will collect different 30K rows without repetition from time_sorted_data dataf
rame
my_final = final[:30000]
my_final.sort_values('Time',inplace=True)
sample = my_final['CleanedText'].values
```

Defining 'WordVector' Class to compute word vectors using TruncatedSVD

In [10]:
```
class WordVector:


    # Initialising the max_features and sample_data to pass in TFIDF vectorizer
    def __init__(self, max_feat , sample_data):
        self.max_feat = max_feat # No.of top words
        self.sample_data = sample_data # document to vectorize
        # List of all top max_feat words
        self.top_words = []
        self.freq = []

    # Picking top max_feat words by using TFIDF vextorizer
    def topWords(self):
        tf_idf_vect = TfidfVectorizer(max_features=self.max_feat)
        tfidf_vec = tf_idf_vect.fit_transform(self.sample_data)
        print("the type of count vectorizer :",type(tfidf_vec))
        print("the shape of out text TFIDF vectorizer : ",tfidf_vec.get_shape())
        print("the number of unique words :", tfidf_vec.get_shape()[1])

        # Top 'n' words
        self.top_words = tf_idf_vect.get_feature_names()
        # tfidf frequencies of top 'n' words
        self.freq = tf_idf_vect.idf_

        return tf_idf_vect.get_feature_names()

    # Computing the co-occurrence matrix with value of neighbourhood as neighbour_n
um
    def cooccurrenceMatrix(self, neighbour_num , list_words):

        # Storing all words with their indices in the dictionary
        corpus = dict()
        # List of all words in the corpus
        doc = []
        index = 0
        for sent in self.sample_data:
            for word in sent.split():
                doc.append(word)
                corpus.setdefault(word,[])
                corpus[word].append(index)
                index += 1

        # Co-occurrence matrix
        matrix = []
        # rows in co-occurrence matrix
        for row in list_words:
            # row in co-occurrence matrix
            temp = []
            # column in co-occurrence matrix
            for col in list_words :
                if( col != row):
                    # No. of times col word is in neighbourhood of row word
                    count = 0
                    # Value of neighbourhood
                    num = neighbour_num
                    # Indices of row word in the corpus
                    positions = corpus[row]
                    for i in positions:
                        if i<(num-1):
                            # Checking for col word in neighbourhood of row
                            if col in doc[i:i+num]:
                                count +=1
                        elif (i>=(num-1)) and (i<=(len(doc)-num)):
                            # Check col word in neighbour of row
```

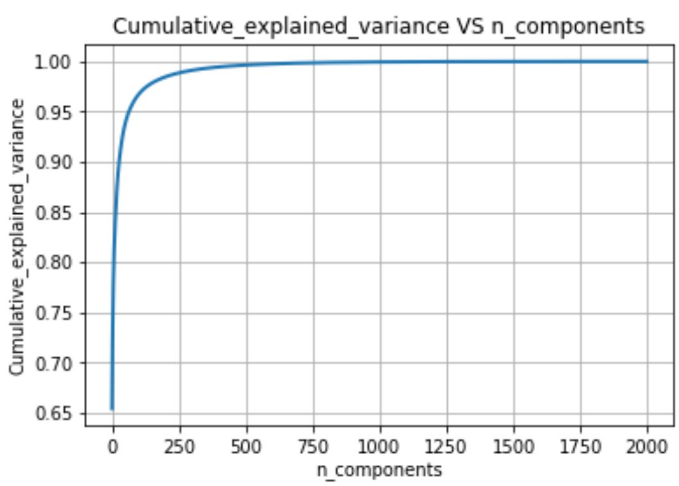
Using WordVector class for computing Word Vectors for top 2K words

In [11]:
```
wv = WordVector(2000,sample)

# Picking top 2K words
words_top = wv.topWords()
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer :  (30000, 2000)
the number of unique words : 2000
```

In [12]:
```
# Computing the co-occurrence matrix for 'words_top' with value of neighbourhood =
5
co_occ_matrix = wv.cooccurrenceMatrix(5, words_top)
print("Shape of co-occurrence matrix : ",co_occ_matrix.shape )
print('\n')

# drawing Cumulative_explained_variance VS n_components plot to find optimal number
of components for co-occurrence matrix
wv.plotCumulativeVariance(co_occ_matrix)
```

```
Shape of co-occurrence matrix :  (2000, 2000)
```

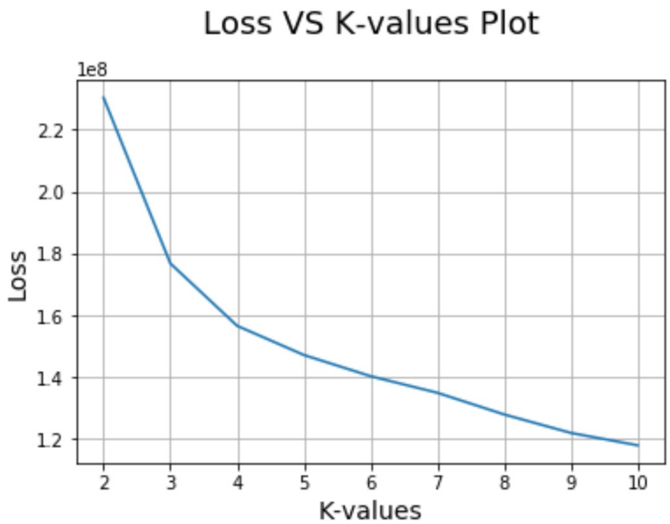


In [13]:
```
# Computing word vectors with 250 components
word_vec_matrix = wv.computeVectors(co_occ_matrix, 250)
print("Shape of word-vector : ",word_vec_matrix.shape)

# Applying k-means with no_of_clusters = 50 on 'word_vec_matrix' and get all cluste
rs
#word_cluster = wv.getClusters(50, word_vec_matrix)
```

```
Shape of word-vector :  (2000, 250)
```

In [14]:
```
from sklearn.cluster import KMeans

k_values = [2,3,4,5,6,7,8,9,10]
loss = []
for i in k_values:
    kmeans = KMeans(n_clusters=i, n_jobs=-1).fit(word_vec_matrix)
    loss.append(kmeans.inertia_)
```

In [15]:
```
plt.plot(k_values, loss)
plt.xlabel('K-values',size=14)
plt.ylabel('Loss',size=14)
plt.title('Loss VS K-values Plot\n',size=18)
plt.grid()
plt.show()
```



In [34]:
```
word_cluster = wv.getClusters(4, word_vec_matrix)
```

In [ ]:

Seeing Words In The Clusters

In [40]:
```
print("Words in Cluster- 1 :\n",word_cluster[0][12:86])
```

```
Words in Cluster- 1 :
 ['address', 'adjust', 'admit', 'adopt', 'ador', 'adult', 'advertis', 'advic', '
advis', 'affect', 'afford', 'afraid', 'afternoon', 'aftertast', 'afterward', 'ag
e', 'agre', 'ahead', 'aid', 'air', 'alcohol', 'aliv', 'allerg', 'allergi', 'allo
w', 'almond', 'alon', 'along', 'alot', 'alreadi', 'altern', 'although', 'america
', 'american', 'among', 'anim', 'answer', 'antioxid', 'anymor', 'anytim', 'anywa
y', 'anywher', 'apart', 'appar', 'appeal', 'appear', 'appetit', 'appl', 'appli',
'appreci', 'appropri', 'approxim', 'area', 'arent', 'aroma', 'aromat', 'artifici
', 'asian', 'asid', 'ask', 'associ', 'assort', 'assum', 'ate', 'attach', 'attemp
t', 'attent', 'attract', 'authent', 'averag', 'avoid', 'aw', 'awar', 'awesom']
```

In [42]:
```
print("Words in Cluster- 2 :\n",word_cluster[2])
```

```
Words in Cluster- 49 :
 ['dog', 'flavor', 'food', 'get', 'good', 'great', 'like', 'love', 'one', 'produ
ct', 'tast', 'tea', 'tri', 'use']
```

In [36]:
```
print("Words in Cluster- 3 :\n",word_cluster[3][27:37])
```

```
Words in Cluster- 49 :
 ['butter', 'came', 'can', 'candi', 'cant', 'care', 'chees', 'chew', 'chicken',
'chocol']
```

SUMMARY :- By observing above cluster we can conclude that words in cluster are related to eatables

Word Clouds

In [38]:
```
wv.generateWordCloud(word_cluster[0])
```

In [39]:
```
wv.generateWordCloud(word_cluster[3])
```

Using WordVector class for computing Word Vectors of top 5K words

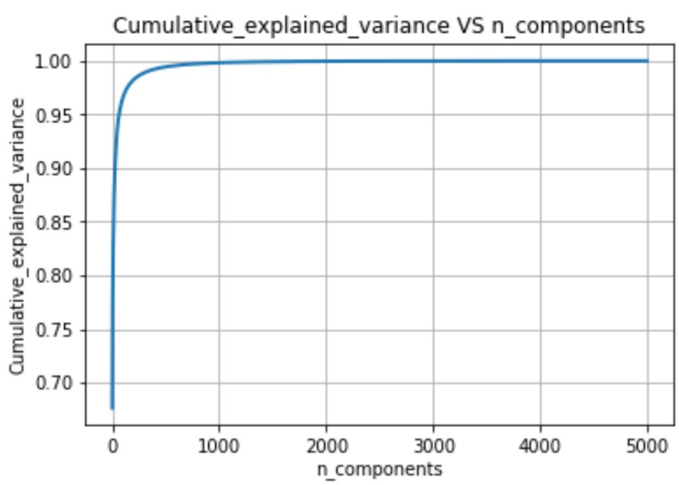
In [ ]:

In [15]:
```
wv1 = WordVector(5000,sample)

# Picking top 5000 words
words_top_5000 = wv1.topWords()
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer :  (30000, 5000)
the number of unique words : 5000
```

In [16]:
```
co_occ_matrix = wv1.cooccurrenceMatrix(5, words_top_5000)
print("Shape of co-occurrence matrix : ",co_occ_matrix.shape )
print('\n')

# drawing Cumulative_explained_variance VS n_components plot to find optimal number of components for co-occurrence matrix
wv1.plotCumulativeVariance(co_occ_matrix)
```

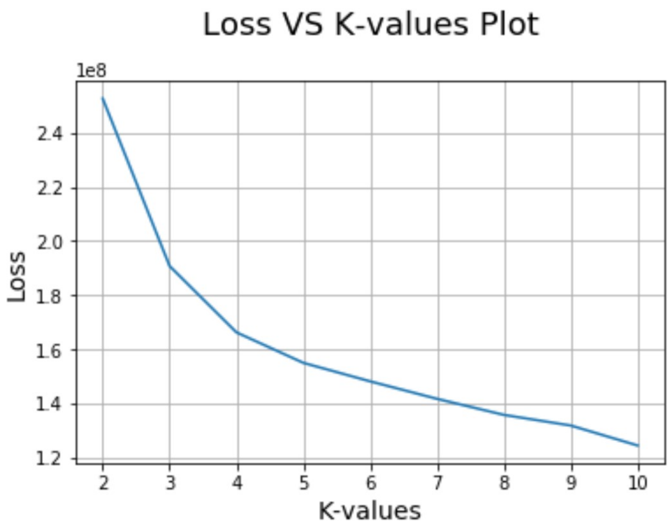
```
Shape of co-occurrence matrix :  (5000, 5000)
```



In [17]:
```
# Computing word vectors with 500 components
word_vec_matrix = wv1.computeVectors(co_occ_matrix, 500)
print("Shape of word-vector : ",word_vec_matrix.shape)

# Applying k-means with no_of_clusters = 50 on 'word_vec_matrix' and get all clusters
#word_cluster = wv1.getClusters(50, word_vec_matrix)
```

```
Shape of word-vector :  (5000, 500)
```

In [18]:
```
from sklearn.cluster import KMeans

k_values = [2,3,4,5,6,7,8,9,10]
loss = []
for i in k_values:
    kmeans = KMeans(n_clusters=i, n_jobs=-1).fit(word_vec_matrix)
    loss.append(kmeans.inertia_)
plt.plot(k_values, loss)
plt.xlabel('K-values',size=14)
plt.ylabel('Loss',size=14)
plt.title('Loss VS K-values Plot\n',size=18)
plt.grid()
plt.show()
```



In [20]:
```
word_cluster = wv1.getClusters(4, word_vec_matrix)
```

In [ ]:

Seeing Words In The Clusters

In [29]:
```
print("Words in Cluster- 2 :\n",word_cluster[2])
```

```
Words in Cluster- 2 :
 ['dog', 'flavor', 'food', 'get', 'good', 'great', 'like', 'love', 'make', 'one'
, 'product', 'tast', 'tea', 'tri', 'use']
```

In [26]:
```
print("Words in Cluster- 1 :\n",word_cluster[0])
```

```
Words in Cluster- 1 :
 ['abil', 'absorb', 'abund', 'abus', 'accent', 'accept', 'access', 'accid', 'acc
ident', 'accompani', 'accomplish', 'accord', 'account', 'accur', 'accustom', 'ac
h', 'achiev', 'acid', 'acknowledg', 'acn', 'acquir', 'acr', 'across', 'act', 'ac
tion', 'activ', 'actor', 'acv', 'adagio', 'adam', 'adapt', 'addict', 'addit', 'a
ddress', 'adequ', 'adher', 'adjust', 'admir', 'admit', 'adopt', 'ador', 'adult',
'advanc', 'advantag', 'adventur', 'advers', 'advertis', 'advic', 'advil', 'advis
', 'affair', 'affect', 'affili', 'afford', 'afghanistan', 'aficionado', 'afraid'
, 'africa', 'african', 'afterlif', 'afternoon', 'aftertast', 'afterward', 'agav'
, 'age', 'agent', 'aggress', 'agre', 'ahead', 'ahi', 'aid', 'aidel', 'ailment',
'aim', 'aint', 'air', 'airlin', 'airport', 'airtight', 'aisl', 'aka', 'akita', '
al', 'ala', 'alarm', 'alaska', 'albeit', 'albertson', 'album', 'alcohol', 'ale',
'alec', 'alert', 'alex', 'alfalfa', 'alfredo', 'alik', 'alittl', 'aliv', 'allerg
', 'allergen', 'allergi', 'allevi', 'allow', 'allsort', 'allspic', 'almond', 'al
o', 'aloha', 'alon', 'along', 'alongsid', 'alot', 'alpo', 'alreadi', 'alright',
'alter', 'altern', 'altogeth', 'altoid', 'aluminum', 'alvita', 'amber', 'america
', 'american', 'amino', 'amish', 'among', 'amongst', 'ampl', 'amus', 'anchovi',
'ancient', 'andes', 'angel', 'angl', 'angri', 'anim', 'anis', 'anni', 'anniversa
ri', 'announc', 'annoy', 'annual', 'answer', 'ant', 'anthocyanin', 'antibiot', '
anticip', 'antioxid', 'anxieti', 'anxious', 'anybodi', 'anymor', 'anytim', 'anyw
ay', 'anywher', 'apart', 'apolog', 'appar', 'appeal', 'appear', 'appet', 'appeti
t', 'appl', 'applesauc', 'appli', 'applic', 'appreci', 'approach', 'appropri', '
approv', 'approx', 'approxim', 'apricot', 'april', 'ara', 'arab', 'arabica', 'ar
ea', 'arent', 'argu', 'arizona', 'arm', 'aroma', 'aromat', 'arrang', 'arrowroot'
, 'art', 'artemi', 'arthriti', 'arthur', 'articl', 'artif', 'artifici', 'artist'
, 'asap', 'ascorb', 'ash', 'asham', 'ashbi', 'asia', 'asian', 'asid', 'ask', 'as
leep', 'asparagus', 'aspartam', 'aspect', 'aspen', 'assam', 'assembl', 'assist',
'associ', 'assort', 'assum', 'assur', 'asthma', 'astonish', 'astronaut', 'ate',
'atleast', 'attach', 'attack', 'attempt', 'attend', 'attent', 'attest', 'attitud
', 'attract', 'attribut', 'augment', 'august', 'aunt', 'aussi', 'australia', 'au
stralian', 'authent', 'author', 'auto', 'automat', 'avenu', 'averag', 'avid', 'a
vocado', 'avoderm', 'avoid', 'aw', 'await', 'awak', 'awar', 'award', 'awesom', '
awhil', 'babi', 'background', 'backpack', 'backyard', 'bacon', 'bacteria', 'bage
l', 'baggi', 'bait', 'baja', 'baker', 'bakeri', 'baklava', 'baklawa', 'balanc',
'baldwin', 'ball', 'balm', 'balsam', 'bam', 'ban', 'banana', 'band', 'bang', 'ba
nk', 'barbara', 'barbecu', 'barbequ', 'bare', 'bargain', 'bariani', 'barista', '
bark', 'barley', 'barn', 'barrel', 'barri', 'basement', 'basi', 'basic', 'basil'
, 'basket', 'bast', 'bat', 'batch', 'bath', 'bathroom', 'batman', 'batter', 'bat
teri', 'battl', 'bay', 'bbq', 'beach', 'beagl', 'beak', 'bear', 'bearabl', 'beas
t', 'beat', 'beaten', 'beauti', 'becam', 'becom', 'becuas', 'bed', 'bedroom', 'b
edtim', 'bee', 'beef', 'beer', 'beet', 'beetlejuic', 'beg', 'began', 'begin', 'b
eginn', 'begun', 'behav', 'behavior', 'behind', 'behold', 'beignet', 'beleiv', '
belgian', 'belgium', 'bell', 'bella', 'belli', 'belong', 'belov', 'bend', 'bene'
, 'benefici', 'benefit', 'bengal', 'bent', 'bergamot', 'berger', 'berri', 'besid
', 'bet', 'beverag', 'bewar', 'beyond', 'bichon', 'bigelow', 'bigger', 'biggest'
, 'bile', 'bill', 'bin', 'bingo', 'birch', 'bird', 'birth', 'birthday', 'biscoff
', 'biscotti', 'biscuit', 'bite', 'bittersweet', 'bizarr', 'blackberri', 'blacke
n', 'bladder', 'blade', 'blah', 'blair', 'blame', 'blanch', 'bland', 'blast', 'b
laze', 'bleach', 'bleed', 'blender', 'bless', 'blew', 'blind', 'bliss', 'bloat',
'block', 'blockag', 'blog', 'blood', 'bloodi', 'bloom', 'blossom', 'blow', 'blow
n', 'blue', 'blueberri', 'board', 'boat', 'bob', 'boba', 'boboli', 'bodi', 'boil
', 'bold', 'bomb', 'bombay', 'bon', 'boneless', 'bonker', 'bonnet', 'bonsai', 'b
onus', 'bonzai', 'boo', 'book', 'boost', 'boot', 'border', 'bore', 'born', 'bors
ari', 'boss', 'boston', 'bother', 'bottom', 'bouillon', 'boulder', 'boullion', '
bounc', 'bound', 'bouquet', 'bourbon', 'bout', 'boutiqu', 'bow', 'bowel', 'bowl'
, 'boxer', 'boy', 'boyfriend', 'boylan', 'bpa', 'brace', 'bragg', 'brain', 'bran
', 'branch', 'brandi', 'brat', 'bratwurst', 'brave', 'bravo', 'brazil', 'break',
'breakag', 'breakfast', 'breast', 'breastf', 'breastfe', 'breastfeed', 'breastmi
lk', 'breath', 'breed', 'breeder', 'breez', 'brewer', 'brick', 'bridg', 'brie',
'brief', 'bright', 'brilliant', 'brine', 'bring', 'british', 'britt', 'brittl',
'broccoli', 'broil', 'broke', 'broken', 'brooklyn', 'broth', 'brother', 'brought
', 'brown', 'browni', 'brows', 'brush', 'brussel', 'btb', 'btw', 'bubbl', 'bubbl
egum', 'buck', 'bucket', 'buckwheat', 'bud', 'buddi', 'budget', 'buffalo', 'buff
et', 'bug', 'bugger', 'build', 'built', 'bulb', 'bulk', 'bull', 'bulldog', 'bull
```

In [27]:
```
print("Words in Cluster- 2 :\n",word_cluster[1])
```

```
Words in Cluster- 2 :
 ['also', 'amazon', 'bag', 'best', 'better', 'bought', 'box', 'brand', 'buy', 'c
at', 'chocol', 'coffe', 'day', 'differ', 'dont', 'drink', 'eat', 'enjoy', 'even'
, 'ever', 'find', 'first', 'found', 'give', 'high', 'hot', 'ive', 'know', 'littl
', 'look', 'mani', 'mix', 'much', 'need', 'never', 'order', 'price', 'purchas',
'realli', 'recommend', 'review', 'sauc', 'say', 'ship', 'sinc', 'store', 'stuff'
, 'thing', 'think', 'time', 'treat', 'two', 'want', 'way', 'well', 'work', 'woul
d', 'year']
```

In [28]:
```
print("Words in Cluster- 4:\n",word_cluster[3])
```

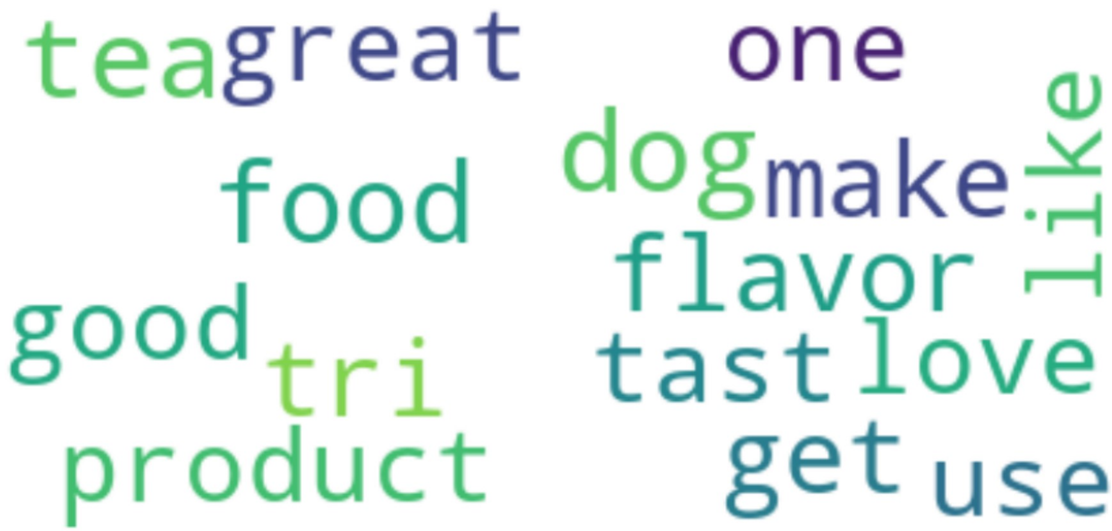
```
Words in Cluster- 4:
 ['abl', 'absolut', 'actual', 'ad', 'add', 'ago', 'almost', 'although', 'alway',
'amaz', 'amount', 'anoth', 'anyon', 'anyth', 'around', 'arriv', 'avail', 'away',
'back', 'bad', 'bake', 'bar', 'base', 'bean', 'believ', 'big', 'bit', 'bitter',
'black', 'blend', 'bone', 'bottl', 'bread', 'brew', 'butter', 'cake', 'call', 'c
alori', 'came', 'can', 'candi', 'cant', 'care', 'carri', 'case', 'chang', 'cheap
er', 'chees', 'chew', 'chicken', 'clean', 'cold', 'color', 'come', 'compani', 'c
ompar', 'contain', 'cook', 'cooki', 'corn', 'cost', 'could', 'couldnt', 'coupl',
'cream', 'cup', 'dark', 'deal', 'decid', 'definit', 'delici', 'didnt', 'diet', '
disappoint', 'dish', 'doesnt', 'dri', 'easi', 'effect', 'egg', 'either', 'els',
'end', 'enough', 'especi', 'everi', 'everyon', 'everyth', 'exact', 'excel', 'exp
ect', 'expens', 'experi', 'extra', 'fact', 'famili', 'far', 'fast', 'favorit', '
feed', 'feel', 'fill', 'fine', 'free', 'fresh', 'friend', 'full', 'gave', 'gift'
, 'glad', 'go', 'goe', 'got', 'green', 'groceri', 'gum', 'half', 'hand', 'happi'
, 'hard', 'havent', 'health', 'healthi', 'heat', 'help', 'home', 'honey', 'hope'
, 'hour', 'hous', 'howev', 'husband', 'ice', 'ill', 'includ', 'ingredi', 'instea
d', 'isnt', 'item', 'jar', 'keep', 'kid', 'kind', 'larg', 'last', 'least', 'leav
', 'less', 'let', 'life', 'light', 'live', 'local', 'long', 'longer', 'lot', 'lo
w', 'made', 'market', 'may', 'mayb', 'meal', 'meat', 'might', 'milk', 'minut', '
money', 'month', 'morn', 'must', 'natur', 'new', 'next', 'nice', 'noth', 'notic'
, 'oil', 'old', 'onlin', 'open', 'organ', 'origin', 'other', 'pack', 'packag', '
past', 'pay', 'peopl', 'pepper', 'per', 'perfect', 'person', 'pet', 'piec', 'pla
ce', 'pleas', 'pod', 'popcorn', 'pound', 'powder', 'prefer', 'pretti', 'probabl'
, 'problem', 'protein', 'puppi', 'put', 'qualiti', 'quick', 'quit', 'read', 'rea
l', 'reason', 'receiv', 'recip', 'red', 'regular', 'rice', 'rich', 'right', 'roa
st', 'run', 'said', 'salad', 'salt', 'save', 'season', 'second', 'see', 'seem',
'sell', 'senseo', 'serv', 'sever', 'size', 'small', 'smell', 'smooth', 'snack',
'someth', 'soup', 'spice', 'spici', 'star', 'start', 'stick', 'still', 'stop', '
strong', 'sugar', 'sure', 'surpris', 'sweet', 'sweeten', 'syrup', 'take', 'tasti
', 'tell', 'textur', 'thank', 'that', 'theyr', 'though', 'thought', 'three', 'to
ok', 'top', 'toy', 'train', 'trap', 'tree', 'turn', 'type', 'usual', 'vanilla',
'varieti', 'vet', 'wasnt', 'water', 'week', 'weight', 'went', 'white', 'whole',
'wish', 'without', 'wonder', 'wont', 'worth', 'yet', 'youll', 'your']
```

Word Clouds

In [25]:
```
wv1.generateWordCloud(word_cluster[0])
```

In [30]:
```
wv1.generateWordCloud(word_cluster[1])
```

In [24]:
```
wv1.generateWordCloud(word_cluster[2])
```



In [31]:
```
wv1.generateWordCloud(word_cluster[3])
```

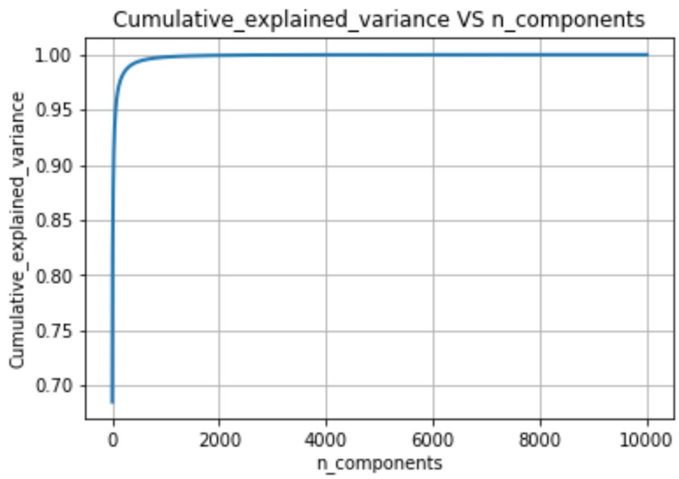
Using WordVector class for computing Word Vectors of top 10K words

In [11]:
```
wv2 = WordVector(10000,sample)

# Picking top 10K words
words_top_10k = wv2.topWords()
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer :  (30000, 10000)
the number of unique words : 10000
```

In [12]:
```
co_occ_matrix = wv2.cooccurrenceMatrix(5, words_top_10k)
print("Shape of co-occurrence matrix : ",co_occ_matrix.shape )
print('\n')

# drawing Cumulative_explained_variance VS n_components plot to find optimal number
of components for co-occurrence matrix
wv2.plotCumulativeVariance(co_occ_matrix)
```

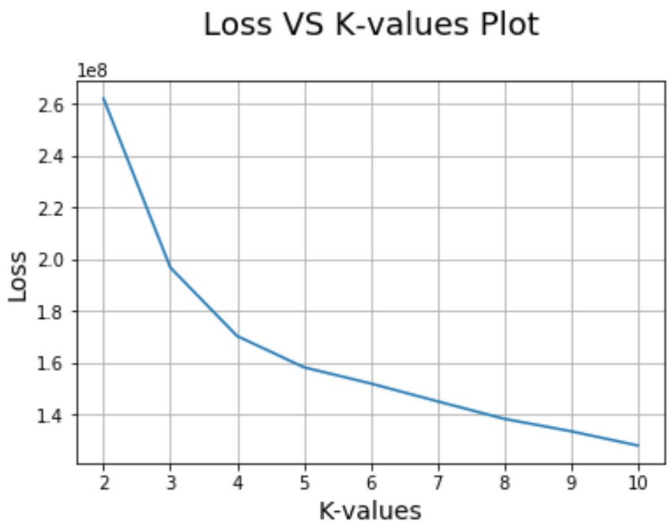
```
Shape of co-occurrence matrix :  (10000, 10000)
```



In [13]:
```
word_vec_matrix = wv2.computeVectors(co_occ_matrix, 1000)
print("Shape of word-vector : ",word_vec_matrix.shape)

# Applying k-means with no_of_clusters = 50 on 'word_vec_matrix' and get all cluste
rs
#word_cluster = wv2.getClusters(50, word_vec_matrix)
```

```
Shape of word-vector :  (10000, 1000)
```

In [14]:
```
from sklearn.cluster import KMeans

k_values = [2,3,4,5,6,7,8,9,10]
loss = []
for i in k_values:
    kmeans = KMeans(n_clusters=i, n_jobs=-1).fit(word_vec_matrix)
    loss.append(kmeans.inertia_)
plt.plot(k_values, loss)
plt.xlabel('K-values',size=14)
plt.ylabel('Loss',size=14)
plt.title('Loss VS K-values Plot\n',size=18)
plt.grid()
plt.show()
```



In [15]:
```
# Applying k-means with no_of_clusters = 50 on 'word_vec_matrix' and get all cluste
rs
word_cluster = wv2.getClusters(4, word_vec_matrix)
```

Seeing Words In The Clusters

In [18]:
```
print("Words in Cluster- 1 :\n",word_cluster[0])
```

```
Words in Cluster- 1 :
 ['aafco', 'aback', 'abandon', 'abbey', 'abbi', 'abc', 'abdomen', 'abdomin', 'ab
il', 'abnorm', 'abomin', 'abound', 'abras', 'abroad', 'abrupt', 'absenc', 'absen
t', 'absolutley', 'absorb', 'absorpt', 'absurd', 'abuelita', 'abund', 'abus', 'a
but', 'acacia', 'academi', 'acai', 'acceler', 'accent', 'accept', 'access', 'acc
essori', 'accid', 'accident', 'acclim', 'accommod', 'accomod', 'accompani', 'acc
omplish', 'accord', 'account', 'accross', 'accumul', 'accur', 'accuraci', 'accus
', 'accustom', 'ace', 'acet', 'aceto', 'ach', 'achey', 'achi', 'achiev', 'achiot
', 'acid', 'acidi', 'acidophilus', 'acknowledg', 'acl', 'acn', 'acquaint', 'acqu
ir', 'acr', 'acrid', 'across', 'act', 'action', 'activ', 'actor', 'acut', 'acv',
'adagio', 'adam', 'adapt', 'addict', 'address', 'adequ', 'adhd', 'adher', 'adhes
', 'adjac', 'adject', 'adjust', 'administ', 'administr', 'admir', 'admit', 'adob
o', 'adolesc', 'adopt', 'ador', 'adorn', 'adren', 'adult', 'adulter', 'adulthood
', 'advanc', 'advantag', 'advent', 'adventur', 'advers', 'advert', 'advertis', '
advic', 'advil', 'advis', 'advisor', 'advoc', 'aerat', 'aero', 'aeropress', 'aes
thet', 'afb', 'affair', 'affect', 'affection', 'afficianado', 'afficionado', 'af
fili', 'affin', 'affirm', 'affix', 'afflict', 'afford', 'afghanistan', 'aficiona
do', 'afraid', 'africa', 'african', 'afteral', 'afterbit', 'afterburn', 'afterli
f', 'aftermath', 'afternoon', 'aftertast', 'afterthought', 'afterward', 'afterwo
rd', 'agar', 'agav', 'age', 'agenc', 'agenda', 'agent', 'aggi', 'aggrav', 'aggre
ss', 'agil', 'agit', 'agoni', 'agre', 'agreeabl', 'agreement', 'agress', 'agricu
ltur', 'ahead', 'ahhh', 'ahi', 'ahmad', 'ahv', 'aid', 'aidel', 'ail', 'ailment',
'aim', 'aint', 'air', 'airborn', 'airedal', 'airfar', 'airi', 'airlin', 'airplan
', 'airport', 'airtight', 'aisl', 'aji', 'ajika', 'ajinomoto', 'aka', 'akc', 'ak
in', 'akita', 'al', 'ala', 'alabama', 'alarm', 'alaska', 'alaskan', 'alba', 'alb
acor', 'albeit', 'albertson', 'album', 'alcohol', 'ale', 'alec', 'alergi', 'aler
t', 'alessi', 'alex', 'alfalfa', 'alfredo', 'alga', 'ali', 'alia', 'alien', 'ali
k', 'alittl', 'aliv', 'alkali', 'alkalin', 'alleg', 'allerg', 'allergen', 'aller
gi', 'allevi', 'alley', 'alli', 'allianc', 'allot', 'allow', 'allsort', 'allspic
', 'allur', 'almighti', 'almond', 'almondi', 'alo', 'aloha', 'alohaisland', 'alo
n', 'along', 'alongsid', 'alot', 'alpha', 'alphabet', 'alpin', 'alpo', 'alreadi'
, 'alright', 'alter', 'altern', 'altho', 'altitud', 'alto', 'altogeth', 'altoid'
, 'alton', 'altruist', 'alum', 'aluminium', 'aluminum', 'alvita', 'alzheim', 'am
a', 'amanda', 'amaranth', 'amaretti', 'amaretto', 'amateur', 'amber', 'ambrosia'
, 'amd', 'amend', 'america', 'american', 'americolor', 'amex', 'ami', 'amino', '
amish', 'amla', 'ammonia', 'ammount', 'among', 'amongst', 'amor', 'amora', 'amou
t', 'amp', 'ampl', 'ampli', 'amplifi', 'amsterdam', 'amus', 'anal', 'analog', 'a
nalysi', 'analyz', 'anastasia', 'ancho', 'anchor', 'anchovi', 'ancient', 'andes'
, 'andi', 'andouill', 'andrew', 'anecdot', 'anem', 'anemia', 'anergen', 'angel',
'angelo', 'anger', 'angl', 'angri', 'anguish', 'anim', 'anis', 'anise', 'anita',
'ankl', 'ann', 'anna', 'annabell', 'annalis', 'annato', 'annatto', 'anni', 'anni
hil', 'anniversari', 'announc', 'annoy', 'annual', 'anonym', 'anorex', 'answer',
'ant', 'antacid', 'anth', 'anthocyanin', 'anti', 'antibacteri', 'antibiot', 'ant
ic', 'antica', 'anticip', 'antico', 'antigua', 'antihistamin', 'antimicrobi', 'a
ntioxid', 'antiqu', 'antirheumat', 'antisept', 'antispasmod', 'antivir', 'antler
', 'antoinett', 'antonio', 'anxieti', 'anxious', 'anya', 'anybodi', 'anyhoo', 'a
nyhow', 'anymor', 'anytim', 'anyway', 'anywher', 'apart', 'ape', 'aperitif', 'ap
ex', 'aphid', 'apiec', 'apnea', 'apo', 'apollo', 'apolog', 'apologet', 'apotheca
ri', 'appal', 'appar', 'appeal', 'appear', 'appet', 'appetit', 'appl', 'applaud'
, 'applesauc', 'applewood', 'appli', 'applianc', 'applic', 'appoint', 'appreci',
'apprehens', 'approach', 'appropri', 'approv', 'approx', 'approxim', 'appx', 'ap
ricot', 'april', 'apso', 'apt', 'aquarium', 'aquir', 'ara', 'arab', 'arabica', '
arar', 'arbol', 'arcan', 'arcana', 'archer', 'archi', 'area', 'arent', 'arepa',
'arf', 'argentina', 'argu', 'arguabl', 'argument', 'aris', 'ariv', 'arizona', 'a
rk', 'arkansa', 'arm', 'armi', 'armour', 'aroma', 'aromat', 'arrang', 'array', '
arrest', 'arriba', 'arrowhead', 'arrowroot', 'arsenal', 'arsenic', 'art', 'artem
i', 'artesian', 'arthrit', 'arthriti', 'arthur', 'artichok', 'articl', 'artif',
'artifici', 'artisan', 'artist', 'artwork', 'arugula', 'asada', 'asap', 'asbach'
, 'ascorb', 'ash', 'asham', 'ashbi', 'ashtray', 'asia', 'asian', 'asid', 'asin',
'ask', 'asleep', 'asparagus', 'aspartam', 'aspca', 'aspect', 'aspen', 'aspergill
us', 'aspertam', 'aspir', 'aspirin', 'ass', 'assam', 'assault', 'assembl', 'asse
rt', 'assess', 'asset', 'assign', 'assist', 'associ', 'assort', 'assum', 'assump
t', 'assur', 'asterisk', 'asthma', 'asthmat', 'astonish', 'astound', 'astragalus
', 'astring', 'astronaut', 'astronom', 'ate', 'ateco', 'athen', 'atherosclerosi'
```

In [19]:
```
print("Words in Cluster- 2 :\n",word_cluster[1])
```

```
Words in Cluster- 2 :
 ['also', 'amazon', 'bag', 'best', 'better', 'bought', 'box', 'brand', 'buy', 'c
at', 'chocol', 'coffe', 'cup', 'day', 'differ', 'dont', 'drink', 'eat', 'enjoy',
'even', 'ever', 'find', 'first', 'found', 'give', 'high', 'hot', 'ive', 'know',
'littl', 'look', 'lot', 'mani', 'mix', 'much', 'need', 'never', 'order', 'price'
, 'purchas', 'qualiti', 'realli', 'recommend', 'review', 'sauc', 'say', 'ship',
'sinc', 'store', 'stuff', 'thing', 'think', 'time', 'treat', 'two', 'want', 'way
', 'well', 'work', 'would', 'year']
```

In [20]:
```
print("Words in Cluster- 3 :\n",word_cluster[2])
```

```
Words in Cluster- 3 :
 ['dog', 'flavor', 'food', 'get', 'good', 'great', 'like', 'love', 'make', 'one'
, 'product', 'tast', 'tea', 'tri', 'use']
```

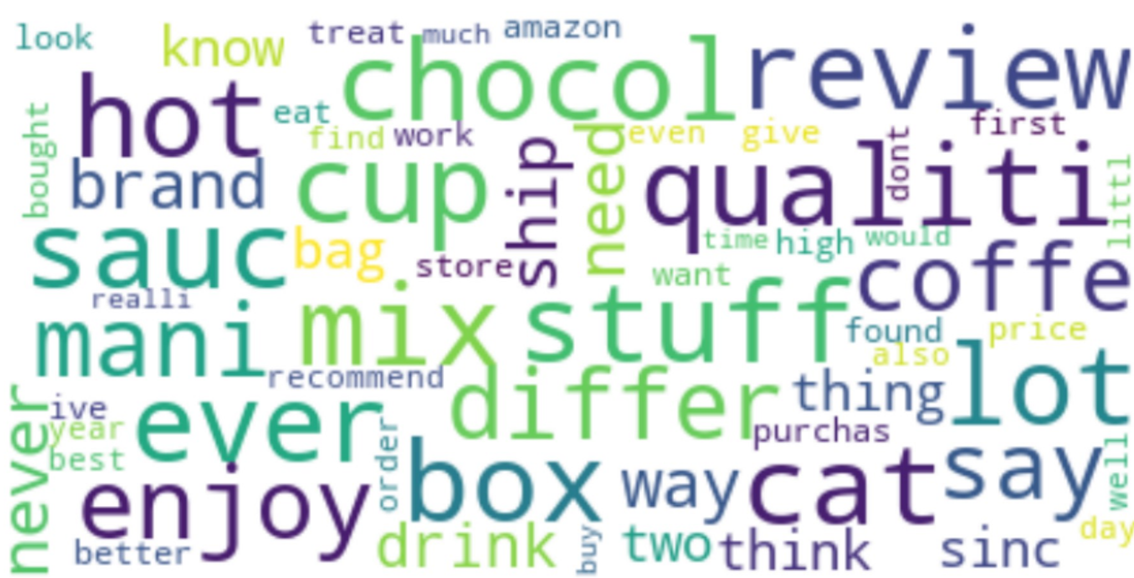
In [21]:
```
print("Words in Cluster- 4 :\n",word_cluster[3])
```

```
Words in Cluster- 4 :
 ['abl', 'absolut', 'actual', 'ad', 'add', 'addit', 'ago', 'almost', 'although',
'alway', 'amaz', 'amount', 'anoth', 'anyon', 'anyth', 'around', 'arriv', 'avail'
, 'away', 'back', 'bad', 'bake', 'bar', 'base', 'bean', 'believ', 'big', 'bit',
'bitter', 'black', 'blend', 'bone', 'bottl', 'bread', 'brew', 'butter', 'cake',
'call', 'calori', 'came', 'can', 'candi', 'cant', 'care', 'carri', 'case', 'chan
g', 'cheaper', 'chees', 'chew', 'chicken', 'christma', 'clean', 'coat', 'cocoa',
'cold', 'color', 'come', 'compani', 'compar', 'contain', 'continu', 'cook', 'coo
ki', 'corn', 'cost', 'could', 'couldnt', 'coupl', 'cream', 'cut', 'dark', 'deal'
, 'decid', 'definit', 'delici', 'didnt', 'diet', 'disappoint', 'dish', 'doesnt',
'dri', 'easi', 'effect', 'egg', 'either', 'els', 'end', 'enough', 'especi', 'esp
resso', 'everi', 'everyon', 'everyth', 'exact', 'excel', 'expect', 'expens', 'ex
peri', 'extra', 'fact', 'famili', 'far', 'fast', 'favorit', 'feed', 'feel', 'fil
l', 'fine', 'fish', 'free', 'fresh', 'friend', 'fruit', 'full', 'gave', 'gift',
'glad', 'go', 'goe', 'got', 'green', 'groceri', 'gum', 'half', 'hand', 'happi',
'hard', 'havent', 'health', 'healthi', 'heat', 'help', 'home', 'honey', 'hope',
'hour', 'hous', 'howev', 'husband', 'ice', 'ill', 'includ', 'ingredi', 'instead'
, 'isnt', 'item', 'jar', 'keep', 'kid', 'kind', 'larg', 'last', 'least', 'leav',
'less', 'let', 'licoric', 'life', 'light', 'list', 'litter', 'live', 'local', 'l
ong', 'longer', 'low', 'made', 'market', 'may', 'mayb', 'meal', 'meat', 'might',
'milk', 'minut', 'money', 'month', 'morn', 'mouth', 'must', 'natur', 'near', 'ne
w', 'next', 'nice', 'night', 'normal', 'noth', 'notic', 'oil', 'old', 'oliv', 'o
nlin', 'open', 'organ', 'origin', 'other', 'pack', 'packag', 'part', 'past', 'pa
y', 'peanut', 'peopl', 'pepper', 'per', 'perfect', 'person', 'pet', 'piec', 'pla
ce', 'pleas', 'plus', 'pod', 'popcorn', 'pound', 'powder', 'prefer', 'pretti', '
probabl', 'problem', 'protein', 'puppi', 'put', 'quick', 'quit', 'read', 'real',
'reason', 'receiv', 'recent', 'recip', 'red', 'regular', 'rememb', 'result', 'ri
ce', 'rich', 'right', 'roast', 'run', 'said', 'salad', 'salt', 'save', 'season',
'second', 'see', 'seem', 'sell', 'senseo', 'serv', 'servic', 'set', 'sever', 'si
de', 'size', 'small', 'smell', 'smooth', 'snack', 'soft', 'someth', 'sometim', '
soup', 'special', 'spice', 'spici', 'star', 'start', 'stick', 'still', 'stock',
'stop', 'strong', 'sugar', 'suppli', 'sure', 'surpris', 'sweet', 'sweeten', 'swi
tch', 'syrup', 'take', 'tasti', 'tell', 'textur', 'thank', 'that', 'theyr', 'tho
ugh', 'thought', 'three', 'took', 'top', 'toy', 'train', 'trap', 'tree', 'turn',
'type', 'usual', 'valu', 'vanilla', 'varieti', 'vet', 'vinegar', 'wasnt', 'water
', 'week', 'weight', 'went', 'white', 'whole', 'wish', 'without', 'wonder', 'won
t', 'worth', 'wouldnt', 'yet', 'yogi', 'youll', 'your']
```

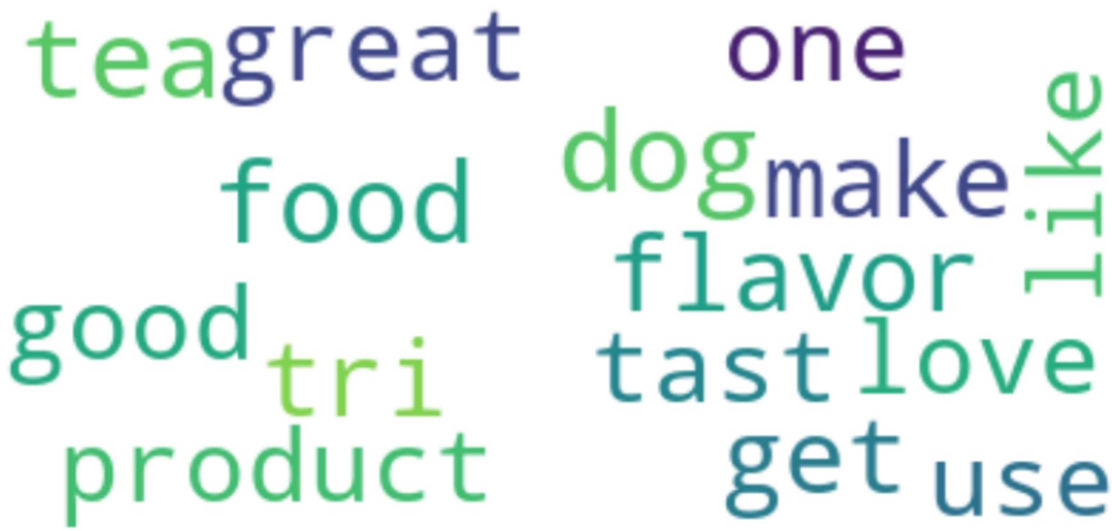
Word Clouds

In [22]:
```
wv2.generateWordCloud(word_cluster[0])
```

In [23]:
```
wv2.generateWordCloud(word_cluster[1])
```



In [24]:
```
wv2.generateWordCloud(word_cluster[2])
```



In [25]:
```
wv2.generateWordCloud(word_cluster[3])
```

In [ ]: