# Amazon Fine Food Reviews Analysis

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

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

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

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - 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 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of dete review.

# [1]. Reading Data

# Mounting Google Drive locally

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189
```

```
Enter your authorization code:
··········
Mounted at /content/gdrive
```

## [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 eff

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefull above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

```
%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
```

```
# using SQLite Table to read data.
con = sqlite3.connect("/content/gdrive/My Drive/Dataset/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

#Using a sample size of 100k datapoints and applyTruncated-SVD on TFIDF
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""",

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative ratin
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 (100000, 10)
```

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Help |
|---|---|---|---|---|---|---|
| **0** | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | |
| **1** | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | |

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



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

```
(80668, 7)
```

| | UserId | ProductId | ProfileName | Time | Score | |
|---|---|---|---|---|---|---|
| **0** | #oc-R115TNMSPFT9I7 | B007Y59HVM | Breyton | 1331510400 | 2 | Overall |

| | | | | | | |
|---|---|---|---|---|---|---|
| **1** | #oc-R11D9D7SHXIJB9 | B005HG9ET0 | Louis E. Emory "hoppy" | 1342396800 | 5 | My wife has |
| **2** | #oc-R11DNU2NBKQ23Z | B007Y59HVM | Kim Cieszykowski | 1348531200 | 1 | This c |
| **3** | #oc-R11O5J5ZVQE25C | B005HG9ET0 | Penguin Chick | 1346889600 | 5 | This w |
| **4** | #oc-R12KPBODL2B5ZD | B007OSBE1U | Christopher P. Presta | 1348617600 | 1 | I d |

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

| | UserId | ProductId | ProfileName | Time | Score | |
|---|---|---|---|---|---|---|
| **80638** | AZY10LLTJ71NX | B006P7E5ZI | undertheshrine "undertheshrine" | 1334707200 | 5 | I was r |

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

```
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 necessa results for the analysis of the data. Following is an example:

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

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Helpfulness |
|---|---|---|---|---|---|---|
| **0** | 78445 | B000HDL1RQ | AR5J8UI46CURR | Geetha Krishnan | 2 | |

| | | | | | |
|---|---|---|---|---|---|
| | | | | Krishnan | |
| **1** | 138317 | B000HDOPYC | AR5J8UI46CURR | Geetha Krishnan | 2 |
| **2** | 138277 | B000HDOPYM | AR5J8UI46CURR | Geetha Krishnan | 2 |
| **3** | 73791 | B000HDOPZG | AR5J8UI46CURR | Geetha | 2 |

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, Helpfulne
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 o

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same produc
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 sin
eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one rep
without sorting would lead to possibility of different representatives still existing for the same product.

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kin
```

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

```
(87775, 10)
```

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

```
87.775
```

**Observation:-** It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than Help
possible hence these two rows too are removed from calcualtions

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

```
display.head()
```

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

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

```
#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()
```

```
(87773, 10)
1    73592
0    14181
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 analysi

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-lett
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

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are
==================================================
```

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore
```

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-fr
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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are
```

```
# 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


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

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore
```

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

```
was way to hot for my blood took a bite and did a jig lol
```

```
# https://gist.github.com/sebleier/554280
```

```
# 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', 'ourselves', 'yo
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll"
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'h
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'unt
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'dur
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'bo
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'ver
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'does
            "hadn't", 'hasn', "hasn't", 'haven', "haven't","isnt", "isn", "isn't", 'ma', 'mig
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
            'won', "won't", 'wouldn', "wouldn't"])


# 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 stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:27<00:00, 3154.02it/s]
```

```
preprocessed_reviews[0]
```

```
'dogs loves chicken product china wont buying anymore hard find chicken products made us
```

```
final["CleanText"] = [preprocessed_reviews[i] for i in range(len(final))]
```

```
final.head(2)
```

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Helpfulnes |
|---|---|---|---|---|---|---|
| **22620** | 24750 | 2734888454 | A13ISQV0U9GZIC | Sandikaye | 1 | |

| | | | | | |
|---|---|---|---|---|---|
| **22621** | 24751 | 2734888454 | A1C298ITT645B6 | Hugh G. | 0 |

```
final_new = final[final.CleanText != '']
```

```
a=0
for qu1 in list(final_new['CleanText']):
    if len(list(qu1))==0:
        print(final_new[final_new['CleanText']== qu1])
        a+=1
print(a)
```

```
0
```

```
len(final)
```

```
87773
```

```
len(final_new)
```

```
87557
```

```
final_new["CleanText"] = [preprocessed_reviews[i] for i in range(len(final_new))]
```

```
final_new.head(2)
```

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Helpfulnes |
|---|---|---|---|---|---|---|
| **22620** | 24750 | 2734888454 | A13ISQV0U9GZIC | Sandikaye | 1 | |
| **22621** | 24751 | 2734888454 | A1C298ITT645B6 | Hugh G. | 0 | |

```
j=0
for i in preprocessed_reviews:
    if i == '':
        j+=1
print(j)
```

```
216
```

```
# Remove empty strings from list of strings
while("" in preprocessed_reviews) :
    preprocessed_reviews.remove("")
```

## [3.2] Preprocessing Review Summary

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

# [4] Featurization

## [4.1] BAG OF WORDS

*1 cell hidden*

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

*1 cell hidden*

## [4.3] TF-IDF

```
tf_idf_vect = TfidfVectorizer(use_idf = True)
tf_idf_vect.fit(final_new["CleanText"].values)
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(final_new["CleanText"].values)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
#print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_sh
```

```
some sample features(unique words in the corpus) ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaa
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (87557, 54842)
```

## [4.4] Word2Vec

*3 cells hidden*

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

#### [4.4.1.1] Avg W2v

*1 cell hidden*

### [4.4.1.2] TFIDF weighted W2v

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# 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_)))


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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████| 4986
```

# Truncated-SVD

Reference Blogs:[blog-1](), [blog-2]()

### [5.1] Taking top features from TFIDF, SET 2

- if the word is occurring more in the corpus then we will get idf value less and
- if we have a rare word in the corpus then we have idf value more.

```
# Get feature names from tfidf
features = tf_idf_vect.get_feature_names()
# feature weights based on idf score
coef = tf_idf_vect.idf_  #The inverse document frequency (IDF) vector
```

```
coef = tf_idf_vect.idf_  #The inverse document frequency (IDF) vector
# Store features with their idf score in a dataframe
co_matrix_df = pd.DataFrame({'Features' : features, 'Idf_score' : coef})
co_matrix_df = co_matrix_df.sort_values("Idf_score", ascending = True)[:3000] #Taking top 300
print("shape of selected features :", co_matrix_df.shape)
print("Top features :\n\n",co_matrix_df[:10])
```

```
shape of selected features : (3000, 2)
Top features :

       Features  Idf_score
32340      not   1.605192
27456     like   2.198294
20395     good   2.312836
20847    great   2.411812
33318      one   2.501118
47789    taste   2.515726
53982    would   2.592148
37616  product   2.652829
28092     love   2.681443
17991   flavor   2.697403
```

## [5.2] Calulation of Co-occurrence matrix

```
#creating an empty a[3000][3000] matrix with all zeros
zero = np.zeros((len(co_matrix_df["Features"].values), len(co_matrix_df["Features"].values)),


#Creating a DataFrame with index = co_matrix_df["Features"].values and columns = co_matrix_df
df_zero = pd.DataFrame(zero, index = co_matrix_df["Features"].values, columns = co_matrix_df[


#Blog 1: https://medium.com/data-science-group-iitr/word-embedding-2d05d270b285
#Blog 2: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
#enumerate() Sendex:https://www.youtube.com/watch?v=bOGmYvtw-kk
#Pandas .loc[] DataSchool:https://www.youtube.com/watch?v=xvpNA7bC8cs
#https://stackoverflow.com/questions/41661801/python-calculate-the-co-occurrence-matrix?nored
#Common Vectorizer usage: https://scikit-learn.org/stable/modules/feature_extraction.html#com
#df_zero:  https://imgur.com/AbyE9rp
#with "window" threshold size we can move forward and backward

%time
window = 4
for sent in tqdm(final_new["CleanText"].values):
    word = sent.split(" ")
    for idx, dumb in enumerate(word):
            for j in range(max(idx-window,0),min(idx+window,len(word))):
                if (word[j] != word[idx]):
                        try:
                            df_zero.loc[word[idx], word[j]] += 1
                            df_zero.loc[word[j], word[idx]] += 1
                        except:
                            pass
print(df_zero)
```

```
print(df_zero)
```

```
  0%|          | 2/87557 [00:00<1:17:17, 18.88it/s]CPU times: user 2 µs, sys: 0 ns, tota
Wall time: 5.72 µs
100%|██████████| 87557/87557 [2:44:22<00:00,  8.88it/s]                 not   like   goo
not                  0  22087  13131   6653  ...       20    31      44     39
```

```
like           22087     0  4332  2481  ...     14    20    29    38
good           13131  4332     0  2496  ...     15    17     8    41
great           6653  2481  2496     0  ...     11     4    11    25
one             9082  3587  2694  1661  ...     32    15    10    18
taste          15163  9787  6385  5470  ...      4     6     0     6
would          12778  4897  3033  1932  ...      8     1    10     7
product         8273  2643  3716  5508  ...      3    10     7    10
love            4587  2011  1659  2282  ...      4     3    33     9
flavor          9828  4378  4066  3922  ...      8     6    26    12
get             7885  1849  1727  1469  ...      4     9     6     8
no              3684  1870  1507  1417  ...      1    56     7     8
really          7570  5371  4801  1827  ...      5     8     6     9
much            8099  3003  1181   916  ...      1     3     9     2
amazon          4091   836  1349  1859  ...      1     6    15     0
time            3828  1065  1198  1147  ...      7     4     3     7
also            3967  2393  2315  2008  ...      1     2     7     4
best            2718  1391   947   773  ...      0     4     7    14
buy             5767  1288  1480  1350  ...      1    13     0     0
little          3065  2059  1308  1223  ...     11     5     6     4
coffee          8544  5833  4093  3387  ...      6     4     0    33
price           3657   852  3576  4065  ...      2    10     2    12
tried           4196  1449   996   647  ...      0     1    16    11
use             4178  1395  1132  1262  ...      3     4     4     7
even            6863  1750  1273   911  ...      5     6     6     2
find            5617  1018  1295  1087  ...      2     0     6     1
well            3919  1358  1356  1159  ...      2     1    10     4
make            4014  1440  1486  1580  ...      3     1     5     6
food            6273  2697  1907  1462  ...     10    32   128     0
try             3843  1722  1176   713  ...     14     7     6    12
...              ...   ...   ...   ...  ...    ...   ...   ...   ...
wood              32    50     2     8  ...      0     0     0     0
mrs               27    20    11    20  ...      0     0     0     0
monster           52    36    10     4  ...      0     0     2     0
tbsp              20    10     2    12  ...      0     0     0     2
acai              57    28    12     4  ...      0     0     0     0
automatically     34    10    12    16  ...      0     0     0     0
grandmother       17    25    16    12  ...      0     0     0     0
safety            56    12     0     5  ...      0     0     0     0
beagle            38    11     4     9  ...      0     0     0     0
bath              24    13     8    21  ...      0     0     0     0
mixer             40     5    13    20  ...      0     0     0     0
plump             50    15    15    26  ...      0     0     0     0
grease            72    23    14     6  ...      0     0     0     0
refuse            46     9     8    11  ...      0     0     0     0
ripped            33    24     9     3  ...      0     0     0     0
ensure            51     7    19     9  ...      0     0     0     0
forced            20    16     4     2  ...      1     0     0     0
stirring          22     6     3    10  ...      0     0     0     0
gummies           63    35    21    21  ...      0     0     0     0
absorb            53    15     5     1  ...      0     0     0     0
notch             21    12    10     8  ...      0     0     0     0
teriyaki          65    51    27    12  ...      0     0     0     0
bills             26     3     9    11  ...      0     0     0     0
issimo            33    12     8    13  ...      0     0     0     3
pretzel           68    38    32    18  ...      0     0     0     0
ruined            62     4    12    10  ...      0     0     0     0
grabbed           20    14    15    11  ...      0     0     0     0
joke              31    20    17     4  ...      0     0     0     0
```

```
    weruva             44      29       8      11  ...         0      0        0       0
    coffe              39      38      41      25  ...         0      0        0       0
```

```
dict = df_zero
file = open('/content/gdrive/My Drive/Dataset/pkcl/c_matrix.csv', 'wb')
pickle.dump(dict, file)
file.close()
```

```
file = open('/content/gdrive/My Drive/Dataset/pkcl/c_matrix.csv', 'rb')
matrix = pickle.load(file)
```

```
matrix.head(2)
```

| | not | like | good | great | one | taste | would | product | love | flavor | get | no |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **not** | 0 | 22087 | 13131 | 6653 | 9082 | 15163 | 12778 | 8273 | 4587 | 9828 | 7885 | 3684 |
| **like** | 22087 | 0 | 4332 | 2481 | 3587 | 9787 | 4897 | 2643 | 2011 | 4378 | 1849 | 1870 |

2 rows × 3000 columns

## [5.3] Finding optimal value for number of components (n) to be retained.

I am trying two approaches here to finding the optimal value for number of components (n)

> ploting cumulative explained variance ratio #Reference: https://www.appliedaicourse.com/lecture/11/applied-course/2896/pca-for-dimensionality-reduction-not-visualization/2/module-2-data-science-exploratory-data-ana

> By using goal level of explained variance #Reference: https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_of_components_in_tsvd/

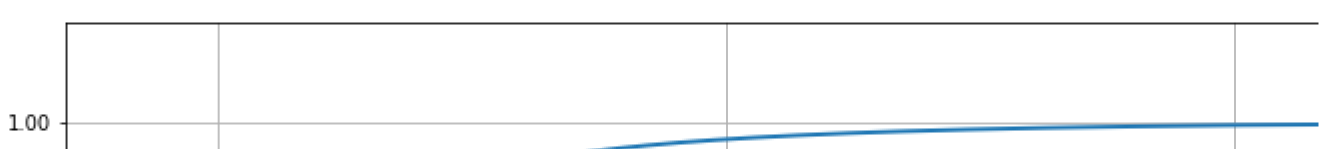
```
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import StandardScaler
from scipy.sparse import csr_matrix
```

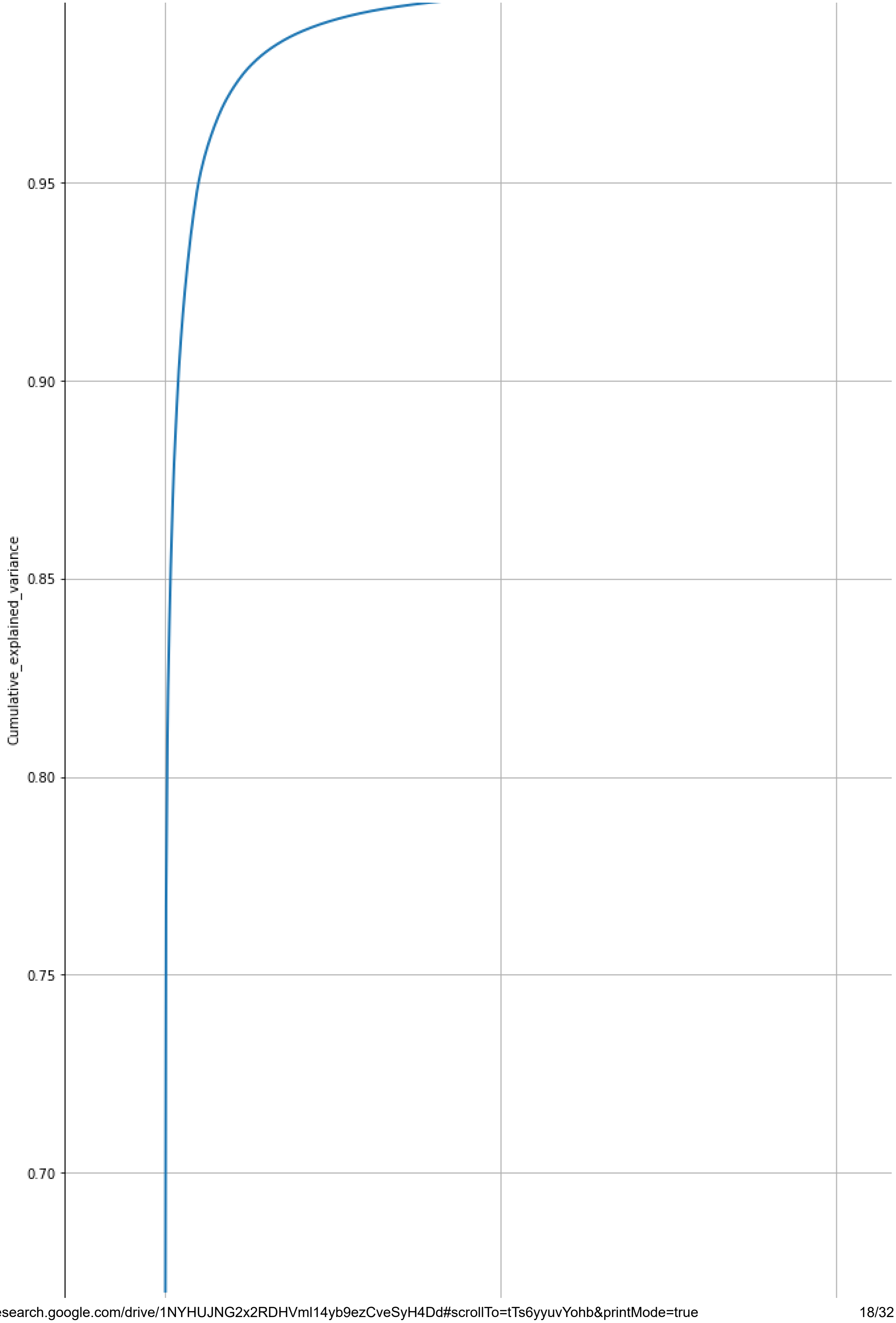
```
from sklearn.random_projection import sparse_random_matrix
```

```
#Reference: https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course
```

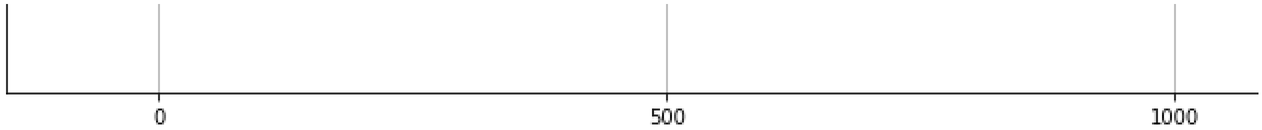
```
# TSVD for dimensionality redcution (non-visualization)
Tsvd = TruncatedSVD(n_components= df_zero.shape[0]-1)
Tsvd.fit(df_zero)
```

```
percentage_var_explained = Tsvd.explained_variance_ / np.sum(Tsvd.explained_variance_)

cum_var_explained = np.cumsum(percentage_var_explained)

# Plot the svd spectrum
plt.figure(1, figsize=(30, 20))

#Tsvd.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```





Cumulative_explained_variance
0.95
0.90
0.85
0.80
0.75
0.70




```
#If I take 650-dimensions, approx. 98% of variance is expalined.
```

By using goal level of explained variance

```
#Reference: https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_of
```

```
#Selecting The Best Number Of Components For TSVD
# Standardize the feature matrix
X = StandardScaler().fit_transform(df_zero)

# Make sparse matrix
X_sparse = csr_matrix(X)
```

```
X_sparse.shape[1]-1
```

```
2999
```

```
#Run Truncated Singular Value Decomposition
# Create and run an TSVD with one less than number of features
tsvd = TruncatedSVD(n_components=X_sparse.shape[1]-1)
X_tsvd = tsvd.fit(X)
```

```
#Create List Of Explained Variances
# List of explained variances
tsvd_var_ratios = tsvd.explained_variance_ratio_
```

```
tsvd_var_ratios
```

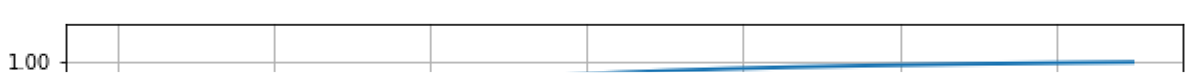
```
array([4.48370511e-01, 2.76818632e-02, 2.12915450e-02, ...,
       2.73328915e-11, 1.65010003e-11, 2.23450257e-12])
```

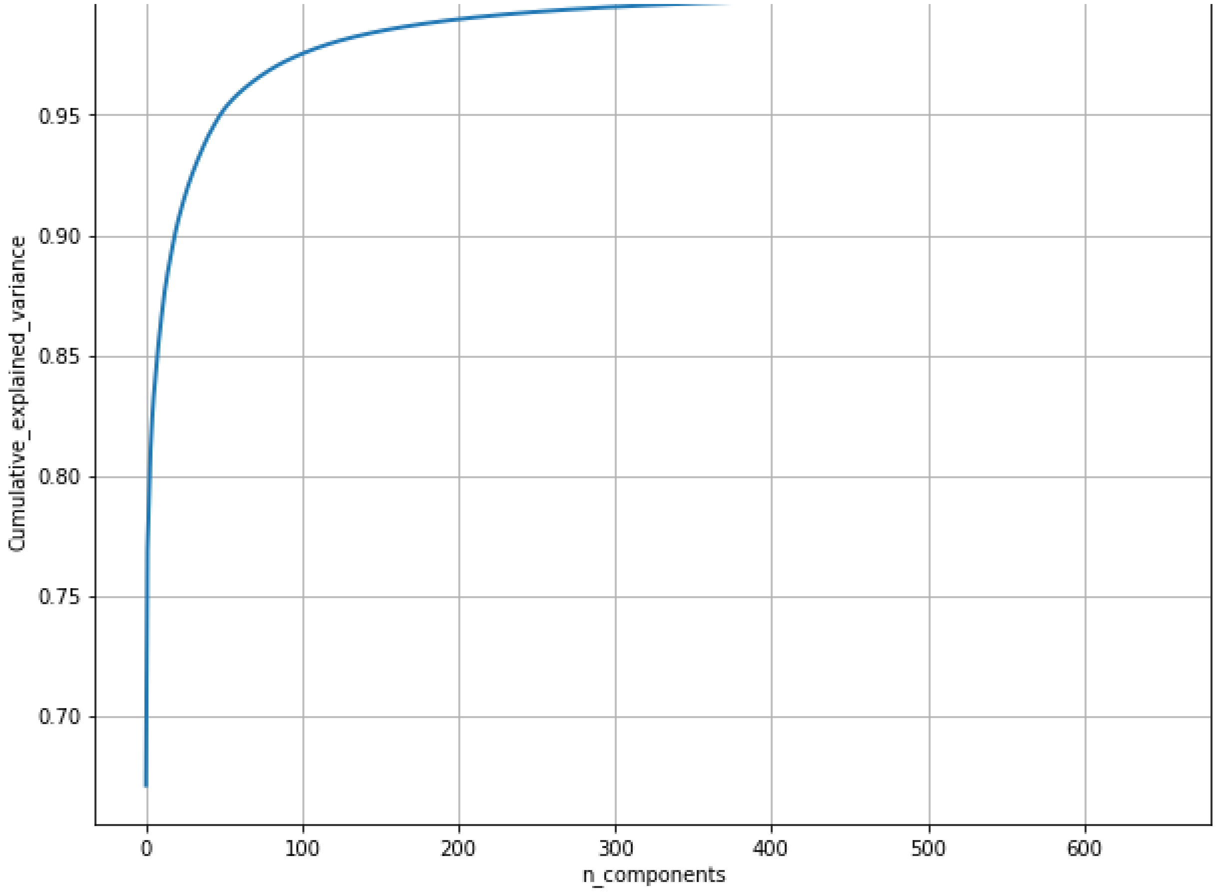
```
#Create Function Calculating Number Of Components Required To Pass Threshold
# Create a function
def select_n_components(var_ratio, goal_var: float) -> int:
    # Set initial variance explained so far
    total_variance = 0.0

    # Set initial number of features
    n_components = 0
```

```
    # For the explained variance of each feature:
    for explained_variance in var_ratio:

        # Add the explained variance to the total
        total_variance += explained_variance

        # Add one to the number of components
        n_components += 1

        # If we reach our goal level of explained variance
        if total_variance >= goal_var:
            # End the loop
            break

    # Return the number of components
    return n_components

# Run function
select_n_components(tsvd_var_ratios, 0.95)
```

```
656
```

```
# TSVD for dimensionality redcution (non-visualization)
Tsvd = TruncatedSVD(n_components= 650)
data_tsvd = Tsvd.fit_transform(df_zero)

percentage_var_explained = Tsvd.explained_variance_ / np.sum(Tsvd.explained_variance_)

cum_var_explained = np.cumsum(percentage_var_explained)

# Plot the svd spectrum
plt.figure(1, figsize=(10, 8))

#Tsvd.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```

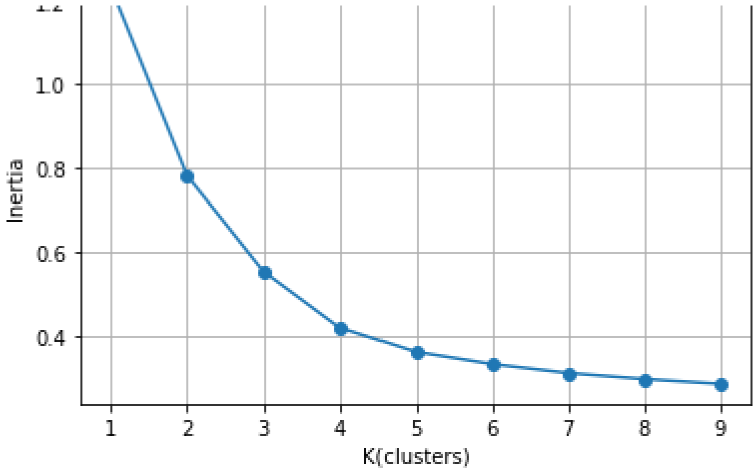





## [5.4] Applying k-means clustering

```
from sklearn.cluster import KMeans


k=range(1,10)
inertia=[]
for i in k:
    model=KMeans(n_clusters=i, n_init=20, n_jobs=-1)
    model.fit(data)
    inertia.append(model.inertia_)
#Calculating "the number of clusters"
n_clusters_ = len(set(model.labels_)) - (1 if -1 in model.labels_ else 0)
print(f"the number of clusters: {n_clusters_}")

#finding best k using elbow method
plt.plot(k, inertia, "-o")
plt.xlabel('K(clusters)')
plt.ylabel('Inertia')
plt.title('Inertia VS K ')
plt.grid()
plt.show()
```

```
the number of clusters: 9
```




```
#https://medium.com/@shritamkumarmund.98/how-dbscan-algorithm-works-2b5bef80fb3
#Computing "the Silhouette Score"
'''A silhouette score ranges from -1 to 1,
with -1 being the worst score possible and
1 being the best score. Silhouette scores of 0 suggest overlapping clusters.'''
print(f"Silhouette Coefficient: {metrics.silhouette_score(data, model.labels_)}")
```

```
Silhouette Coefficient: 0.5856666602061852
```

## [5.5] Wordclouds of clusters obtained in the above section

```
Best_Features = [str(i) for i in co_matrix_df["Features"]]
```

```
clusters = []
for i in list(set(model.labels_)):
    words = []
    for word in range(model.labels_.shape[0]):
        if (model.labels_[word] == i):
            words.append(Best_Features[word])
    clusters.append(words)
```

```
import pickle
import matplotlib.pyplot as plt
```

```
dict = clusters
file = open('/content/gdrive/My Drive/Dataset/pkcl/clusters.txt', 'wb')
pickle.dump(dict, file)
file.close()
```

```
#saving the clusters in a path. If something went wrong and RAM got crassed then I don't need
file = open('/content/gdrive/My Drive/Dataset/pkcl/clusters.txt', 'rb')
clusters = pickle.load(file)
```

```
cluster_1 = list()
cluster_2 = list()
cluster_3 = list()
cluster_4 = list()
cluster_5 = list()
cluster_6 = list()
cluster_7 = list()
cluster_8 = list()
cluster_9 = list()

for i in range(len(clusters)):
    if i==0:
        for n in clusters[i]:
            cluster_1.append(n)

    elif i ==1:
        for n in clusters[i]:
            cluster_2.append(n)

    elif i ==2:
        for n in clusters[i]:
            cluster_3.append(n)

    elif i ==3:
        for n in clusters[i]:
            cluster_4.append(n)

    elif i ==4:
        for n in clusters[i]:
            cluster_5.append(n)

    elif i ==5:
        for n in clusters[i]:
            cluster_6.append(n)

    elif i ==6:
        for n in clusters[i]:
            cluster_7.append(n)

    elif i ==7:
        for n in clusters[i]:
            cluster_8.append(n)

    else:
        for n in clusters[i]:
            cluster_9.append(n)
```

```
print(f"cluster_1 has {len(cluster_1)} no. of points")
print(f"cluster_2 has {len(cluster_2)} no. of points")
print(f"cluster_3 has {len(cluster_3)} no. of points")
print(f"cluster_4 has {len(cluster_4)} no. of points")
print(f"cluster_5 has {len(cluster_5)} no. of points")
print(f"cluster_6 has {len(cluster_6)} no. of points")
print(f"cluster_7 has {len(cluster_7)} no. of points")
print(f"cluster_8 has {len(cluster_8)} no. of points")
print(f"cluster_9 has {len(cluster_9)} no. of points")
```

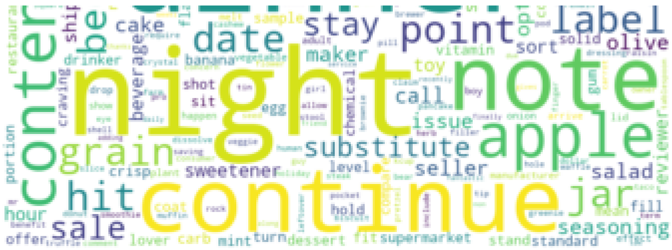
```
cluster_1 has 2465 no. of points
cluster_2 has 2 no. of points
cluster_3 has 1 no. of points
cluster_4 has 6 no. of points
cluster_5 has 31 no. of points
cluster_6 has 126 no. of points
cluster_7 has 1 no. of points
cluster_8 has 2 no. of points
cluster_9 has 366 no. of points
```

```
#https://stackoverflow.com/questions/16645799/how-to-create-a-word-cloud-from-a-corpus-in-pyt

#for cluster_1
print(f"Word cloud for cluster_1, words present {len(cluster_1)}")
data=''
for i in cluster_1:
    data += " " + i
from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=400,background_color='white',stopwords = {}).generate
# Display the wordcloud image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
#for cluster_2
print(f"Word cloud for cluster_2, words present {len(cluster_2)}")
data=''
for i in cluster_2:
    data += " " + i
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white",stopwords = {}).generate(data)
# Display the wordcloud image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
#for cluster_3
print(f"Word cloud for cluster_3, words present {len(cluster_3)}")
data=''
for i in cluster_3:
    data += " " + i
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white",stopwords = {}).generate(data)
```

```
# Display the wordcloud image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
#for cluster_4
print(f"Word cloud for cluster_4, words present {len(cluster_4)}")
data=''
for i in cluster_4:
    data += " " + i
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white",stopwords = {}).generate(data)
# Display the wordcloud image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

#for cluster_5
print(f"Word cloud for cluster_5, words present {len(cluster_5)}")
data=''
for i in cluster_5:
    data += " " + i
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white",stopwords = {}).generate(data)
# Display the wordcloud image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

#for cluster_6
print(f"Word cloud for cluster_6, words present {len(cluster_6)}")
data=''
for i in cluster_6:
    data += " " + i
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white",stopwords = {}).generate(data)
# Display the wordcloud image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

#for cluster_7
print(f"Word cloud for cluster_7, words present {len(cluster_7)}")
data=''
for i in cluster_7:
    data += " " + i
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white",stopwords = {}).generate(data)
# Display the wordcloud image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

```
#for cluster_8
print(f"Word cloud for cluster_8, words present {len(cluster_8)}")
data=''
for i in cluster_8:
    data += " " + i
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white",stopwords = {}).generate(data)
# Display the wordcloud image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

#for cluster_9
print(f"Word cloud for cluster_9, words present {len(cluster_9)}")
data=''
for i in cluster_9:
    data += " " + i
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white",stopwords = {}).generate(data)
# Display the wordcloud image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

```
Word cloud for cluster_1, words present 2465
```



Word cloud for cluster_2, words present 2


Word cloud for cluster_3, words present 1

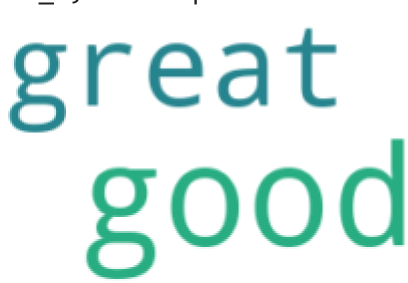

Word cloud for cluster_4, words present 6


Word cloud for cluster_5, words present 31

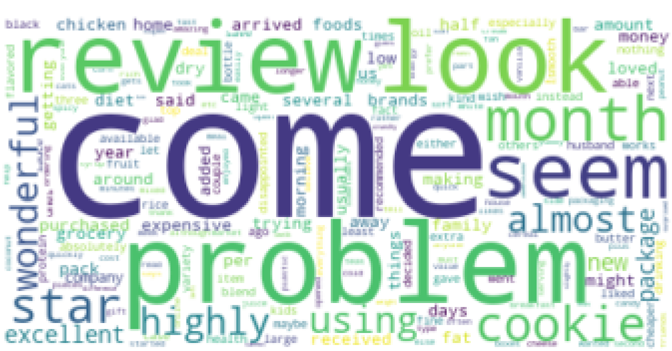

```
Word cloud for cluster_6, words present 126
```


```
Word cloud for cluster_7, words present 1
```


```
Word cloud for cluster_8, words present 2
```




```
Word cloud for cluster_9, words present 366
```




**Observation:** In cluster_2 wordcloud reprentation we can see the words like 'great' and 'good', in Cluster_4 we have v have words like "Store" and "Bag". This indicate that all the word are sensible in each clusters. That means the dimer

## [5.6] Function that returns most similar words for a given word.

```
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity
from sklearn.metrics.pairwise import cosine_similarity


from nltk.stem.snowball import SnowballStemmer
from prettytable import PrettyTable


#https://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html
#https://stackoverflow.com/questions/18424228/cosine-similarity-between-2-number-lists

from numpy import linalg

def cosine_similarity(value1, value2):

    dotprod = np.dot(value1, value2)
    normprod = linalg.norm(value1) * linalg.norm(value2)

    cosin_dist = dotprod/normprod
    cosin_sim = 1 - cosin_dist

    return (cosin_sim)


def returns_most_similar_words(word):

    sno = SnowballStemmer(language='english')
    input_word=(sno.stem(word.lower()))
    top_words=list(co_matrix_df["Features"])

    index = None

    if word not in top_words:
        print(f"Word '{word}' is not present.")

    else:
        print(f"For word '{word}' most similar words: ")
        for i in range(len(top_words)):
            if input_word == top_words[i]:
                index = i

        similarity_values = []
        for i in range(data_tsvd.shape[0]):
            similarity_values.append(cosine_similarity(data_tsvd[i], data_tsvd[index]))

        similarity_values= np.array(similarity_values)
        sorted_index = similarity_values.argsort()
```

```
    similarity_words = []
    similarity_scores = []
    for i in range(1, 11):
        similarity_words.append(top_words[sorted_index[i]])
        simscore = 1 - similarity_values[sorted_index[i]]
        similarity_scores.append(simscore)

    table = PrettyTable()
    table.add_column("Similar Words", similarity_words)
    table.add_column("Similarity Scores", similarity_scores)
    print(table)
```

```
returns_most_similar_words("tea")
```

```
For word 'tea' most similar words:
+---------------+--------------------+
| Similar Words | Similarity Scores  |
+---------------+--------------------+
|     coffee    | 0.8301753149476128 |
|      also     | 0.8111822276450024 |
|     still     | 0.7984751211465887 |
|      teas     | 0.7949693316969705 |
|     flavor    | 0.7912622682283627 |
|     course    |  0.78670185355107  |
|      one      | 0.7862521065713114 |
|     olives    | 0.7858296610722836 |
|     really    | 0.7832637730165124 |
|      fact     | 0.7824875122556161 |
+---------------+--------------------+
```

```
returns_most_similar_words("like")
```

```
For word 'like' most similar words:
+---------------+--------------------+
| Similar Words | Similarity Scores  |
+---------------+--------------------+
|       ok      | 0.9376718387972346 |
|      good     | 0.9217353395738282 |
|      okay     | 0.9210781252247653 |
|      true     | 0.9172397505304584 |
|    terrible   | 0.9166586374829859 |
|    although   | 0.9159440879894366 |
|     liking    | 0.9130205735600214 |
|    however    | 0.910591390780306  |
|    perhaps    | 0.9092507965002709 |
|      bad      | 0.9091276529724434 |
+---------------+--------------------+
```

```
returns_most_similar_words("order")
```

```
For word 'order' most similar words:
+---------------+--------------------+
| Similar Words | Similarity Scores  |
+---------------+--------------------+
```

```
|    ordering   | 0.9452310870605548 |
|    purchase   | 0.9288570585402267 |
|      buy      | 0.9089952014740637 |
|    ordered    | 0.9030958673423635 |
|      item     | 0.8965523159058947 |
|   purchasing  | 0.8939019735502052 |
|     buying    | 0.8851537532023238 |
|    reorder    | 0.8812893815751778 |
|    receive    | 0.8787435566604812 |
|     offer     | 0.8757428559124876 |
+---------------+--------------------+
```

```
returns_most_similar_words("product")
```

```
For word 'product' most similar words:
+---------------+--------------------+
| Similar Words | Similarity Scores  |
+---------------+--------------------+
|      also     | 0.9035119149083028 |
|      item     | 0.9012054714027643 |
|     stuff     |  0.89901542969623  |
|     still     | 0.8895611661715822 |
|     anyway    | 0.8856121941023972 |
|     course    | 0.8845083900225769 |
|    however    | 0.8840023416704055 |
|     think     | 0.8774229896974832 |
|       us      | 0.8770602351073469 |
|   packaging   | 0.873432140708761  |
+---------------+--------------------+
```

```
returns_most_similar_words("shritam")
```

```
Word 'shritam' is not present.
```

# [6] Conclusions

```
from prettytable import PrettyTable
x= PrettyTable()
y= PrettyTable()

x.field_names = ["Dimensionality reduction technique", "n_components"]
y.field_names = ["Vectorizer", "Model", "No. of Clusters", "Silhouette Coefficient"]

x.add_row(["TruncatedSVD", 650])
y.add_row(["TFIDF", "K-Means", 9, 0.5856666602061852])

print(x)
print()

print("Appled K-means:")
print(y)
```

| Dimensionality reduction technique | n_components |
| --- | --- |
| TruncatedSVD | 650 |

Appled K-means:

| Vectorizer | Model | No. of Clusters | Silhouette Coefficient |
| --- | --- | --- | --- |
| TFIDF | K-Means | 9 | 0.5856666602061852 |