```
from google.colab import drive


drive.mount("/content/drive/")
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mo
```

```
pip install pyforest
```

```
Collecting pyforest
  Downloading https://files.pythonhosted.org/packages/8f/85/77a9d2d9ff240822039f3dfd7a3
Building wheels for collected packages: pyforest
  Building wheel for pyforest (setup.py) ... done
  Created wheel for pyforest: filename=pyforest-1.0.3-py2.py3-none-any.whl size=13715 s
  Stored in directory: /root/.cache/pip/wheels/af/0b/39/340a7f15fc8d4ff5ab50847b28789af
Successfully built pyforest
Installing collected packages: pyforest
Successfully installed pyforest-1.0.3
```

```
import pyforest


lazy_imports()
```

```
```

```
    ['import seaborn as sns',
     'import plotly.express as px',
     'from sklearn.feature_extraction.text import TfidfVectorizer',
     'from sklearn.ensemble import GradientBoostingClassifier',
     'from sklearn.manifold import TSNE',
     'from sklearn.ensemble import RandomForestRegressor',
     'import datetime as dt',
     'import re',
     'import lightgbm as lgb',
     'import altair as alt',
     'import spacy',
     'import nltk',
     'import sklearn',
     'import pickle',
     'from pyspark import SparkContext',
     'import dash',
     'import pydot',
     'from pathlib import Path',
     'import sys',
     'import tensorflow as tf',
     'import pandas as pd',
     'import tqdm',
     'from sklearn.ensemble import GradientBoostingRegressor',
     'import bokeh',
     'import xgboost as xgb',
     'import matplotlib.pyplot as plt',
     'from dask import dataframe as dd',
     'import numpy as np',
     'from openpyxl import load_workbook',
     'from sklearn import svm',
     'import plotly.graph_objs as go',
     'import gensim',
     'import os',
     'import plotly as py',
     'import statistics',
     'from sklearn.preprocessing import OneHotEncoder',
     'import awswrangler as wr',
     'import keras',
     'from sklearn.model_selection import train_test_split',
     'import matplotlib as mpl',
     'from sklearn.ensemble import RandomForestClassifier',
     'import glob']


import sqlite3

con=sqlite3.connect("/content/drive/My Drive/colab folder/Datasets/amazon-fine-food-reviews/

con

 <sqlite3.Connection at 0x7f7f0627f3b0>

database=pd.read_sql_query("select * from reviews where score<>3 limit 20000",con)


database
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulne
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	
...	...	...	...	...	...	
19995	21784	B000KV61FC	A3FVKI0UH9DO2A	S. Malosh	1	
19996	21785	B000KV61FC	A3ACVJEAM4L2LQ	Elb	1	
19997	21786	B000KV61FC	AHHWZ4723VGOL	Sherry Lynn	1	
19998	21787	B000KV61FC	A2O4CZ102I8Q2K	jus42day	1	
19999	21788	B000KV61FC	A3I4GCI6XTX1BB	Eric C. Vizinas "Q"	1	

20000 rows × 10 columns

```
def partition(x):
  if x>3:
    return 1
  return 0

database.Score=database.Score.map(partition)

database.Score.value_counts()
```

```
1    16855
0     3145
Name: Score, dtype: int64
```

```
database=database[database.HelpfulnessNumerator<=database.HelpfulnessDenominator]
sort=database.sort_values("ProductId")
final=sort.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inp
final.shape
```

```
(19354, 10)
```

```
final.Score.value_counts()
```

```
1    16339
0     3015
Name: Score, dtype: int64
```

```
from bs4 import BeautifulSoup
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
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'y
            "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'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"])
```

```
%%time
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in 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())
```

```
CPU times: user 6.31 s, sys: 141 ms, total: 6.45 s
Wall time: 6.45 s
```

```
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].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_summary.append(sentance.strip())
```

```
100%|██████████| 19354/19354 [00:04<00:00, 4628.82it/s]
```

```
final["Cleaned_text"]=preprocessed_reviews
final["Cleaned_summary"]=preprocessed_summary
final["Combined_text"]=final.Cleaned_text.values+" "+final.Cleaned_summary.values
```

```
/usr/local/lib/python3.6/dist-packages/pyforest/__init__.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  # -*- coding: utf-8 -*-
/usr/local/lib/python3.6/dist-packages/pyforest/__init__.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  from ._imports import *
/usr/local/lib/python3.6/dist-packages/pyforest/__init__.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  from .utils import (
```

```
def tfidf(x_train):
    # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
  model = TfidfVectorizer()
  list_of_sentance=[]
  for sentance in x_train:
    list_of_sentance.append(sentance.split())

  w2v_models=Word2Vec(list_of_sentance,size=50,min_count=10,workers=8)
  w2v_words=list(w2v_models.wv.vocab)

  tf_idf_matrix = model.fit_transform(x_train)
  # we are converting a dictionary with word as a key, and the idf as a value
  dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
  # 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 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_models.wv[word]
              #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
              # to reduce the computation we are
              # dictionary[word] = idf value of word in whole courpus
              # sent.count(word) = tf valeus of word in this review
              tf_idf = dictionary[word]*(sent.count(word)/len(sent))
              sent_vec += (vec * tf_idf)
              weight_sum += tf_idf
      if weight_sum != 0:
          sent_vec /= weight_sum
      tfidf_sent_vectors.append(sent_vec)
      row += 1
  return np.array(tfidf_sent_vectors)


def avg_words(x_train):
  i=0
  list_of_sentance=[]
  for sentance in x_train:
    list_of_sentance.append(sentance.split())

  w2v_models=Word2Vec(list_of_sentance,size=50,min_count=10,workers=8)
  w2v_words=list(w2v_models.wv.vocab)

  sent_vectors = []

  for sent in list_of_sentance:
      sent_vec = np.zeros(50)
      cnt_words =0;
      for word in sent: #
          if word in w2v_words:
              vec = w2v_models.wv[word]
              sent_vec += vec
              cnt_words += 1
      if cnt_words != 0:
          sent_vec /= cnt_words
      sent_vectors.append(sent_vec)

  return np.array(sent_vectors)



from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer


from gensim.models import Word2Vec


X=final.Combined_text


count_vec=CountVectorizer(dtype="float",max_features=1000,min_df=10)
```

```
count_vec.fit(X)
```

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype='float', encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=1000, min_df=10,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

```
bow_feature=count_vec.get_feature_names()
```

```
bow_X=count_vec.transform(X)
```

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

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

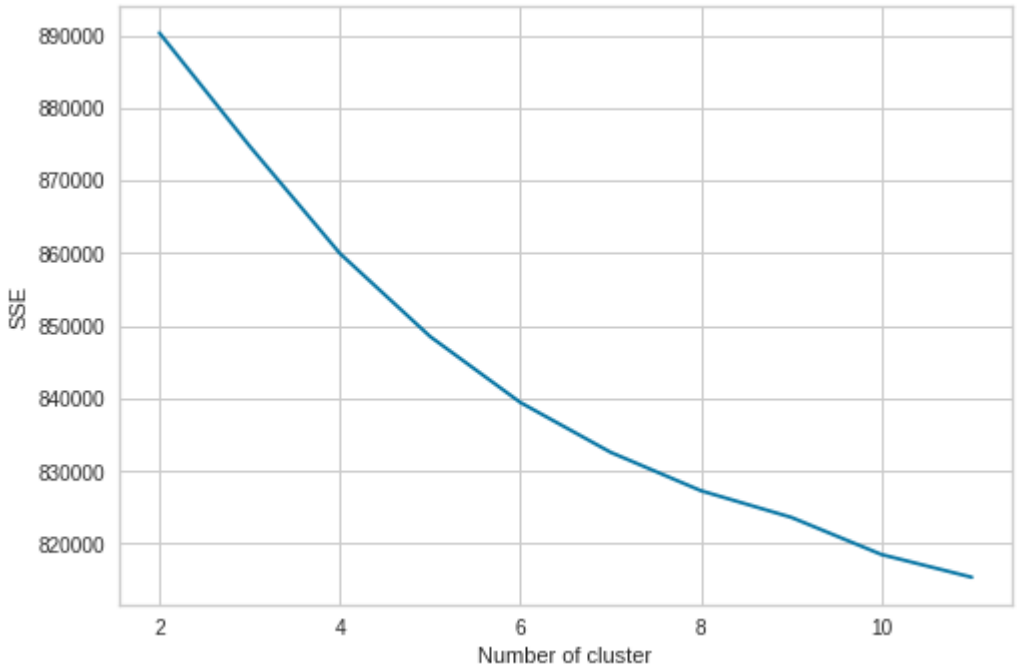
```
bow_X.shape
```

```
(19354, 1000)
```

```
from yellowbrick.cluster import KElbowVisualizer
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning:
  warnings.warn(message, FutureWarning)
```

```
%%time
sse_bow={}
for i in range(2,12):
  k_bow=KMeans(n_clusters=i,n_jobs=-1,max_iter=1000)
  k_bow.fit(bow_X)
  sse_bow[i] = k_bow.inertia_
plt.figure()
plt.plot(list(sse_bow.keys()), list(sse_bow.values()))
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
plt.show()
```




```
CPU times: user 602 ms, sys: 145 ms, total: 748 ms
Wall time: 29min 29s
```

```
sse_bow
```

```
{2: 890311.8287845688,
 3: 874723.6569852777,
 4: 859945.6464110438,
 5: 848508.8330072237,
 6: 839428.3938382778,
 7: 832569.7837259754,
 8: 827283.5351045557,
 9: 823620.8106735769,
 10: 818499.5080713605,
 11: 815372.9462227848}
```

```
%%time
k_bow=KMeans(n_clusters=4,max_iter=1000,n_jobs=-1)
bow_predicted=k_bow.fit_predict(bow_X)
bow_predicted
```

```
CPU times: user 61.2 ms, sys: 53.4 ms, total: 115 ms
Wall time: 2min 54s
```

```
bow_cluster_center=k_bow.cluster_centers_
print(bow_cluster_center)
```

```
[[0.04033187 0.03894907 0.01590228 ... 0.00944918 0.01152339 0.02973035]
 [0.02201071 0.02870315 0.00349494 ... 0.00646936 0.02104402 0.03889054]
 [0.11743119 0.05504587 0.06055046 ... 0.02018349 0.00917431 0.04770642]
 [0.03033268 0.02935421 0.02837573 ... 0.00195695 0.00880626 0.01369863]]
```

```
bow_predicted.shape
```

```
(19354,)
```

```
bow_X.todense()
```

```
    matrix([[0., 0., 0., ..., 0., 0., 0.],
            [0., 0., 0., ..., 0., 0., 0.],
            [0., 0., 0., ..., 0., 0., 0.],
            ...,
            [0., 0., 0., ..., 0., 0., 0.],
            [0., 0., 0., ..., 0., 0., 0.],
            [0., 0., 0., ..., 0., 0., 0.]])
```

```
pip install wordcloud
```

```
Requirement already satisfied: wordcloud in /usr/local/lib/python3.6/dist-packages (1.5
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from w
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.6/dist-packages (
```

```
from wordcloud import WordCloud
```

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt

def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        max_words=200,
        max_font_size=40,
        scale=3,
        random_state=1 # chosen at random by flipping a coin; it was heads
    ).generate(str(data))

    fig = plt.figure(1, figsize=(12, 12))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()
```

```
k_bow.labels_
```

```
array([1, 1, 1, ..., 3, 0, 1], dtype=int32)
```

```
k_bow.labels_[5]
```

```
1
```

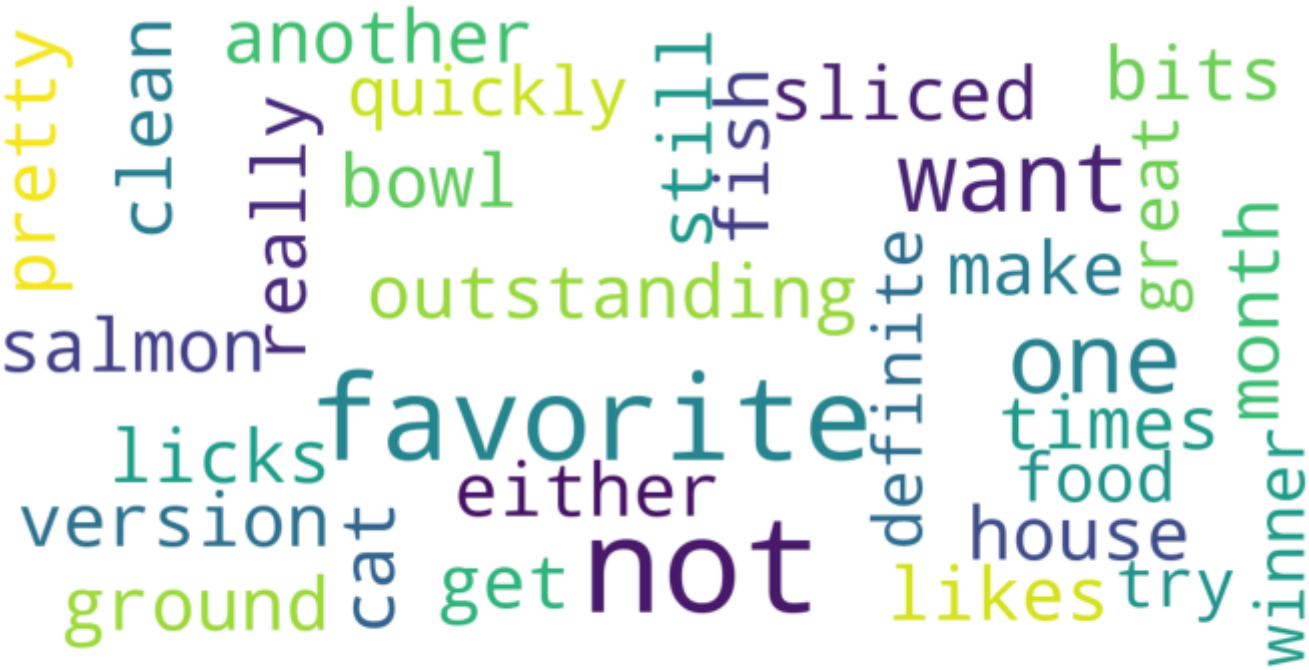
```
# getting original text i.e review column from the dataset
text_reviews = final["Combined_text"].values
cluster_1 = []
cluster_2 = []
cluster_3 = []
cluster_4 = []
for i in range(k_bow.labels_.shape[0]):
    if k_bow.labels_[i] == 0:
        cluster_1.append(text_reviews[i])
    elif k_bow.labels_[i] == 1:
```

```
        cluster_2.append(text_reviews[i])
    elif k_bow.labels_[i] == 2:
        cluster_3.append(text_reviews[i])
    else:
        cluster_4.append(text_reviews[i])
```

```
cluster_1[0]
```

```
'another favorite house cat not want times month still wants licks bowl clean fish not
```

```
show_wordcloud(cluster_1[0],title="bag of words of KMeans")
```



bag of words of KMeans

```
tf_vec=TfidfVectorizer(max_features=1000,min_df=10,dtype="float")
```

```
tf_vec.fit(X)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/feature_extraction/text.py:1817: UserWar

    Only (<class 'numpy.float64'>, <class 'numpy.float32'>, <class 'numpy.float16'>) 'dtype

    TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                    dtype='float', encoding='utf-8', input='content',
                    lowercase=True, max_df=1.0, max_features=1000, min_df=10,
                    ngram_range=(1, 1), norm='l2', preprocessor=None,
                    smooth_idf=True, stop_words=None, strip_accents=None,
                    sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                    tokenizer=None, use_idf=True, vocabulary=None)
```

```
X.shape
```

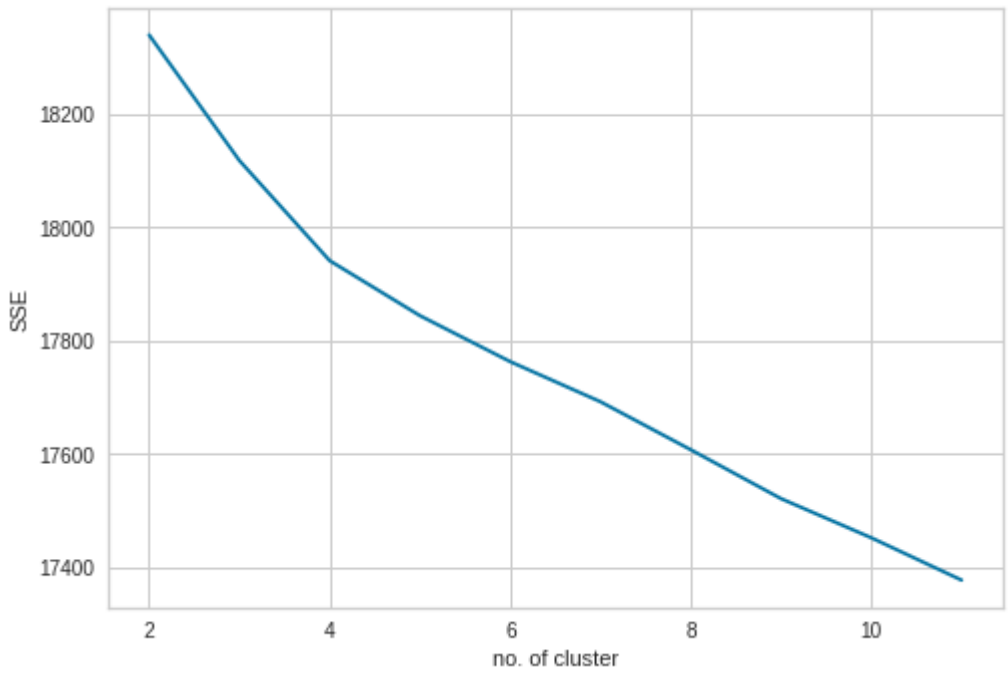
```
(19354,)
```

```
tfidf_X=tf_vec.transform(X)
```

```
tfidf_X.shape
```

```
(19354, 1000)
```

```
%%time
sse_tfidf={}
for i in range(2,12):
  k_tfidf=KMeans(n_clusters=i,max_iter=1000,n_jobs=-1)
  k_tfidf.fit(tfidf_X)
  sse_tfidf[i]=k_tfidf.inertia_
plt.figure()
plt.plot(list(sse_tfidf.keys()),list(sse_tfidf.values()))
plt.xlabel("no. of cluster")
plt.ylabel("SSE")
plt.show()
```




```
CPU times: user 635 ms, sys: 143 ms, total: 778 ms
Wall time: 33min 26s
```

```
sse_tfidf
```

```
{2: 18338.9277561945,
 3: 18117.525202304412,
 4: 17941.034512925777,
 5: 17844.07125457114,
 6: 17763.048431300245,
 7: 17692.371941994403,
 8: 17607.774481643555,
 9: 17521.577069050938,
 10: 17452.55820480163,
 11: 17377.922227030318}
```

```
%%time
k_tfidf=KMeans(n_clusters=4,max_iter=1000,n_jobs=-1)
tfidf_predict=k_tfidf.fit_predict(tfidf_X)
```

```
CPU times: user 81.9 ms, sys: 71.4 ms, total: 153 ms
Wall time: 2min 12s
```

```
print(k_tfidf.labels_)
print(tfidf_predict)
print(k_tfidf.cluster_centers_)
```

```
[1 1 1 ... 3 3 1]
[1 1 1 ... 3 3 1]
[[0.00498562 0.00458977 0.00143516 ... 0.00012976 0.0014602  0.00289021]
 [0.00471536 0.00520781 0.00102182 ... 0.00197327 0.00535212 0.00918913]
 [0.00453581 0.00831074 0.0004311  ... 0.0003851  0.00169457 0.00270352]
 [0.00427988 0.00394    0.00503561 ... 0.00018408 0.00277648 0.00384048]]
```

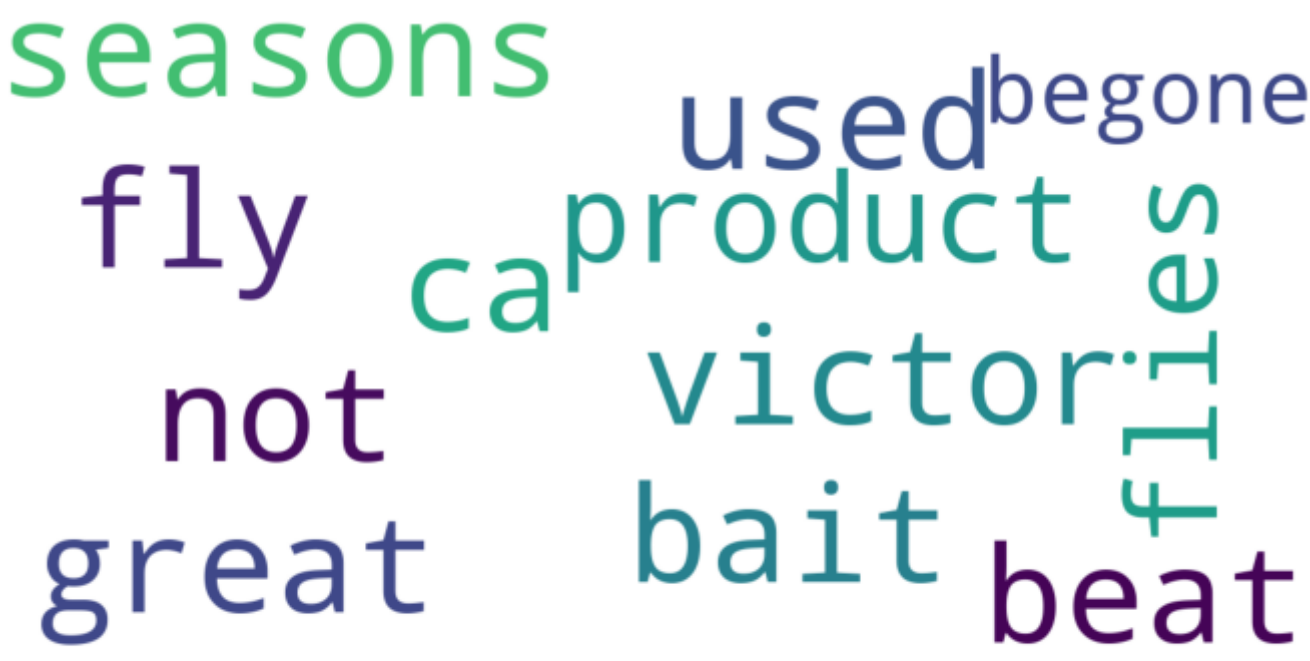
```
text=X.values
tfidf_cluster0=[]
tfidf_cluster1=[]
tfidf_cluster2=[]
tfidf_cluster3=[]
for i in range(k_tfidf.labels_.shape[0]):
  if k_tfidf.labels_[i]==0:
    tfidf_cluster0.append(text[i])
  elif k_tfidf.labels_[i]==1:
    tfidf_cluster1.append(text[i])
  elif k_tfidf.labels_[i]==2:
    tfidf_cluster2.append(text[i])
  else:
    tfidf_cluster3.append(text[i])
```

```
print(len(tfidf_cluster0))
print(len(tfidf_cluster1))
print(len(tfidf_cluster2))
print(len(tfidf_cluster3))
```

```
1058
13611
2629
2056
```

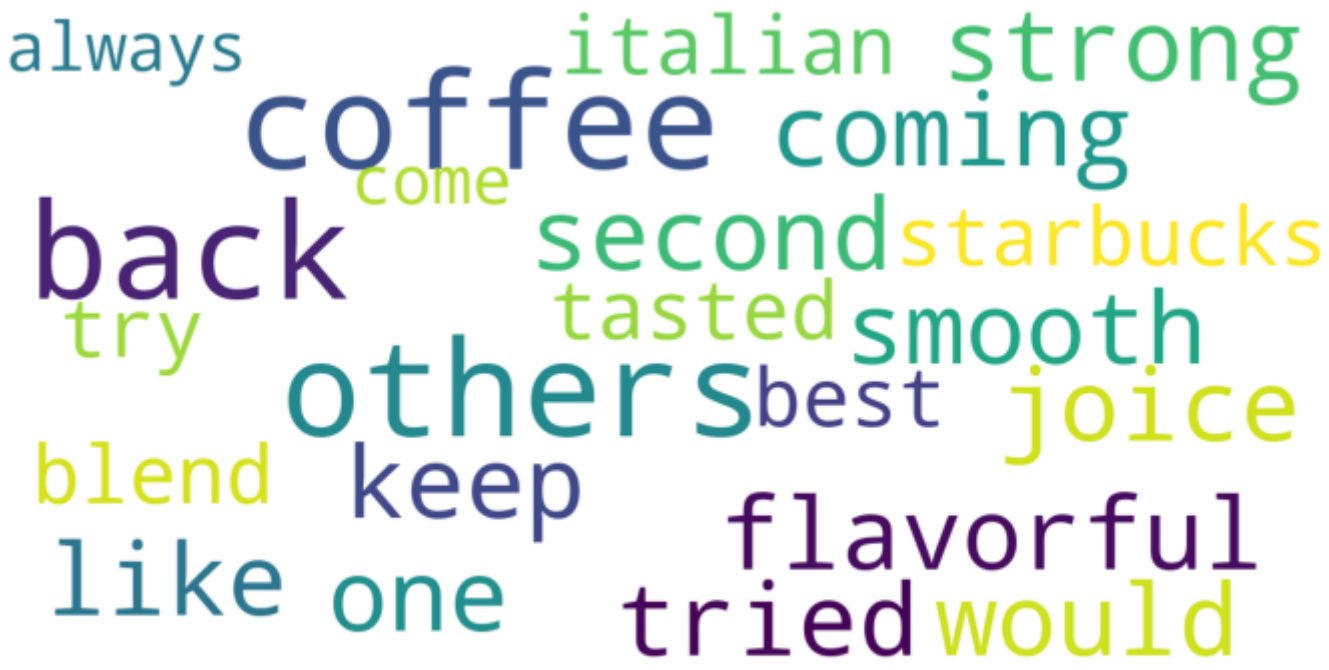
```
show_wordcloud(tfidf_cluster0[0],title="tfidf of KMeans of cluster 0")
show_wordcloud(tfidf_cluster1[0],title="tfidf of KMeans of cluster 1")
show_wordcloud(tfidf_cluster2[0],title="tfidf of KMeans of cluster 2")
show_wordcloud(tfidf_cluster3[0],title="tfidf of KMeans of cluster 3")
```

tfidf of KMeans of cluster 0



tfidf of KMeans of cluster 1

tfidf of KMeans of cluster 2

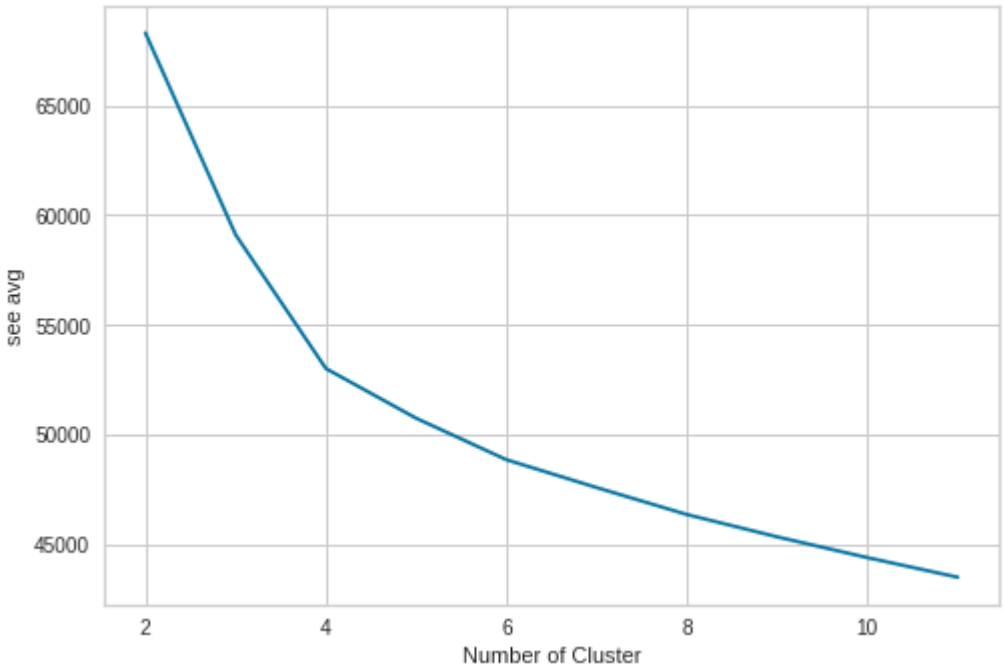


tfidf of KMeans of cluster 3

```
avg_w2v=avg_words(X)
```

```
weighted_tfidf=tfidf(X)
```

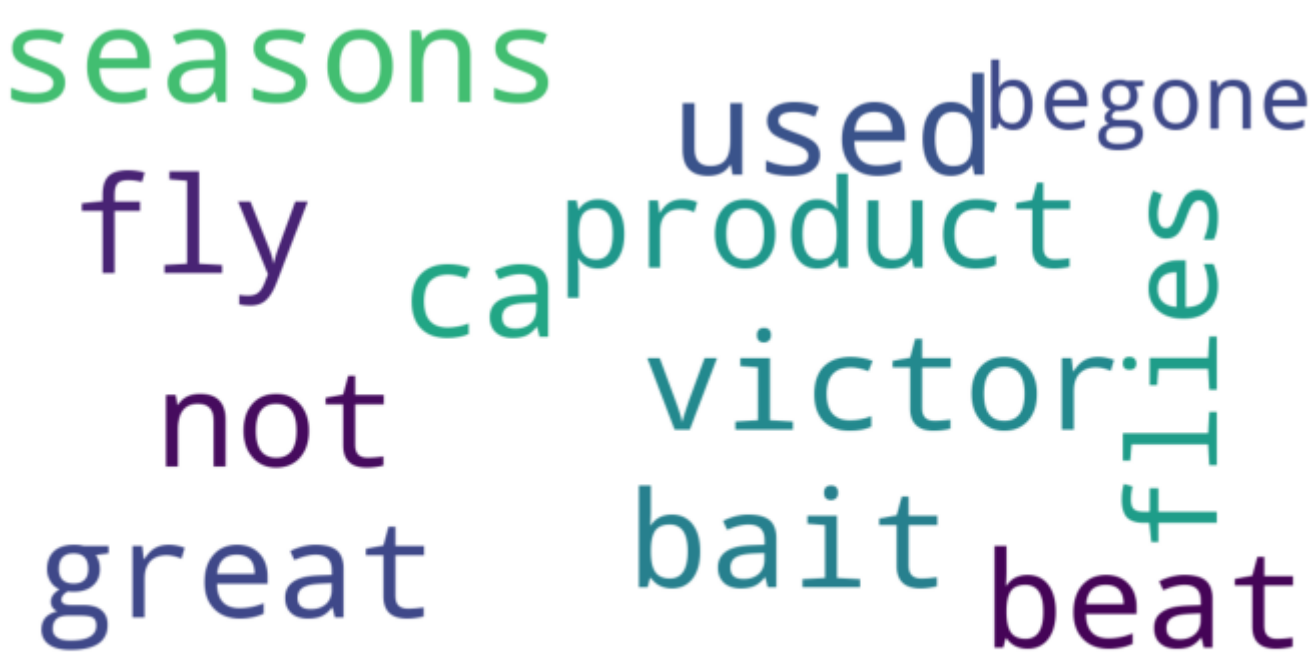
```
sse_avg_w2v={}
for i in range(2,12):
  k_avg=KMeans(n_clusters=i,n_jobs=-1,max_iter=1000)
  k_avg.fit(avg_w2v)
  sse_avg_w2v[i]=k_avg.inertia_
plt.plot(list(sse_avg_w2v.keys()),list(sse_avg_w2v.values()))
plt.xlabel("Number of Cluster")
plt.ylabel("see avg")
plt.show()
```




```
%%time
k_avg=KMeans(n_clusters=4,max_iter=1000,n_jobs=-1)
k_avg.fit(avg_w2v)
print(k_avg.labels_)
print(k_avg.cluster_centers_)
```

```
    [0 0 0 ... 1 1 0]
    [[ 0.38507036 -0.04766366  0.18808399 -0.05136491 -0.50221201 -0.31174853
      -0.21924399  0.67627108  0.05549016 -0.01409827  0.1533457   0.47452051
      -0.05046334  0.13525134 -0.04402118 -0.08617691  0.04267859  0.21967979
       0.12024228 -0.27690708 -0.03861204 -0.43239804  0.26024121 -0.00505421
      -0.21482515 -0.37960811 -0.06646985  0.57244702 -0.33093834  0.02268404
      -0.41111487 -0.35537955 -0.05958295 -0.03806612  0.02372729 -0.26886052
       0.35786476  0.18545371 -0.18720212 -0.12921072 -0.02862135 -0.39366144
      -0.10400658  0.51176916  0.70786001 -0.46423635  0.72999814  0.10064552
       0.14732313 -0.22140184]
     [ 0.26366932  0.06365385  0.35415379  0.3085838   0.08364429 -0.32257074
      -0.56983373  0.71814329 -0.0652635   0.56220935 -0.27415965  0.6803123
       0.6176473   0.30364086  0.04905281 -0.14497054  0.12534331  0.8045418
       0.02199762 -0.22951496  0.03908797 -0.42387333  0.1623559  -0.35355192
      -0.216957   -0.71188254 -0.20979306  0.81679931 -0.54211258  0.20093179
      -0.75230921 -0.53063365  0.03689195  0.04612071 -0.06801149 -0.32483259
       0.5545067   0.38521113 -0.12480815  0.22793165  0.36966106 -0.5110883
      -0.72157718  0.88693352  0.10977166 -0.36250819  0.25511977 -0.21440578
       0.32989331 -0.35408282]
     [ 0.44576153  0.07615372  0.00689272 -0.25935015 -0.52686435 -0.49629169
      -0.20697821  0.59132156  0.28619972 -0.55486279  0.53879225  0.24092109
      -0.25255128 -0.09599087 -0.18611652 -0.00680114  0.3141864   0.35845712
      -0.11924901  0.15832666 -0.09049019 -0.68292396  0.48949882 -0.00982469
       0.09483278 -0.32944038 -0.26322586  0.14798486 -0.54549187 -0.08838907
      -0.07351766 -0.07658877  0.09326899 -0.234187    0.05942827 -0.24045838
       0.21002265 -0.43660126 -0.15564571 -0.16862222 -0.49260137 -0.65069468
       0.21173211  0.3227021   0.27916842 -0.72237823  0.86081083  0.43097086
       0.72783285 -0.10814798]
     [ 0.48181164  0.10245996  0.39059036  0.02185205 -0.06926073 -0.11645913
      -0.50342659  0.72753206  0.26164416 -0.09282425 -0.00889655  0.03183609
      -0.08266034 -0.04824056  0.00422422 -0.16662987  0.19986958  0.16879102
       0.09958558  0.01489475 -0.19075921 -0.42966934  0.50660315  0.01213157
      -0.04384955 -0.66425085 -0.09161025  0.47322296 -0.40397583  0.10388358
      -0.29681122 -0.39046146  0.23684209 -0.22137647  0.06660256  0.24358533
       0.27828535  0.420309   -0.53913387 -0.03899227 -0.02794952 -0.54302619
      -0.31663614  0.41369636  0.28441631 -0.3886182   0.36445404 -0.28858978
       0.32096884 -0.21814216]]
    CPU times: user 73.4 ms, sys: 100 ms, total: 174 ms
    Wall time: 1.94 s
```

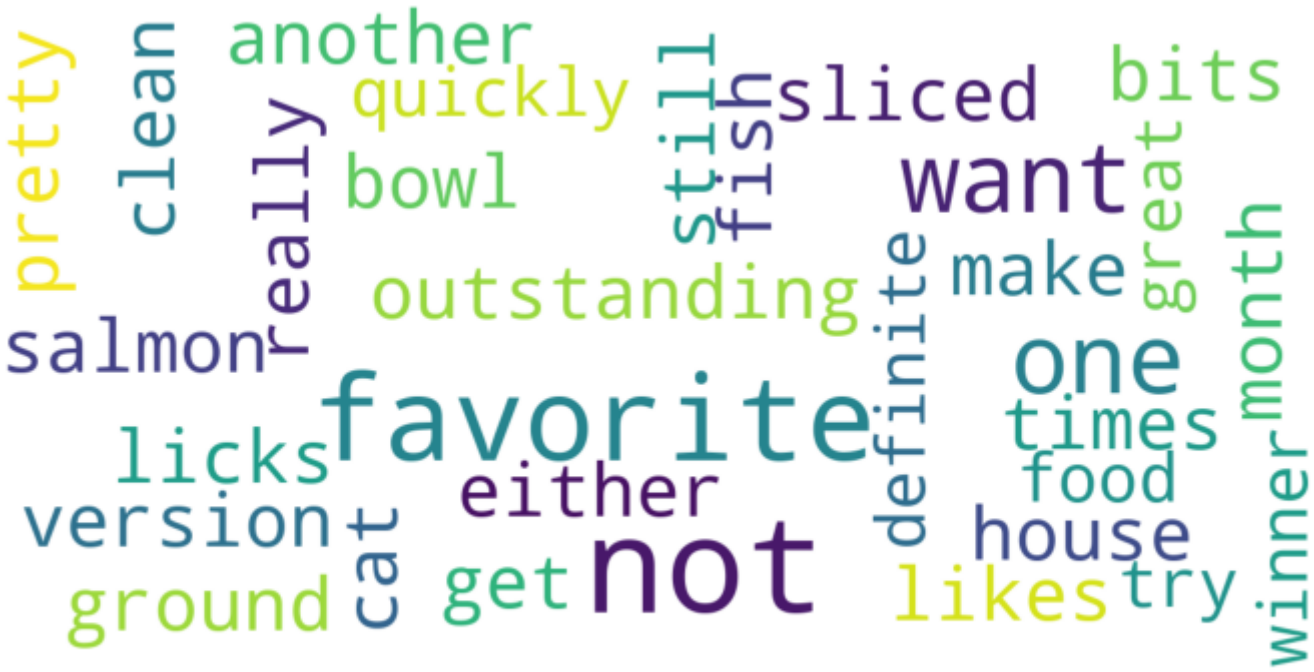
```
avg_cluster0=[]
avg_cluster1=[]
avg_cluster2=[]
avg_cluster3=[]
for i in range(k_avg.labels_.shape[0]):
  if k_avg.labels_[i]==0:
    avg_cluster0.append(text[i])
  elif k_avg.labels_[i]==1:
    avg_cluster1.append(text[i])
  elif k_avg.labels_[i]==2:
    avg_cluster2.append(text[i])
  else:
    avg_cluster3.append(text[i])


show_wordcloud(avg_cluster0[0],title="avg_w2v of KMeans of cluster 0")
show_wordcloud(avg_cluster1[0],title="avg_w2v of KMeans of cluster 1")
show_wordcloud(avg_cluster2[0],title="avg_w2v of KMeans of cluster 2")
show_wordcloud(avg_cluster3[0],title="avg_w2v of KMeans of cluster 3")
```



avg_w2v of KMeans of cluster 0

avg_w2v of KMeans of cluster 1

avg_w2v of KMeans of cluster 2

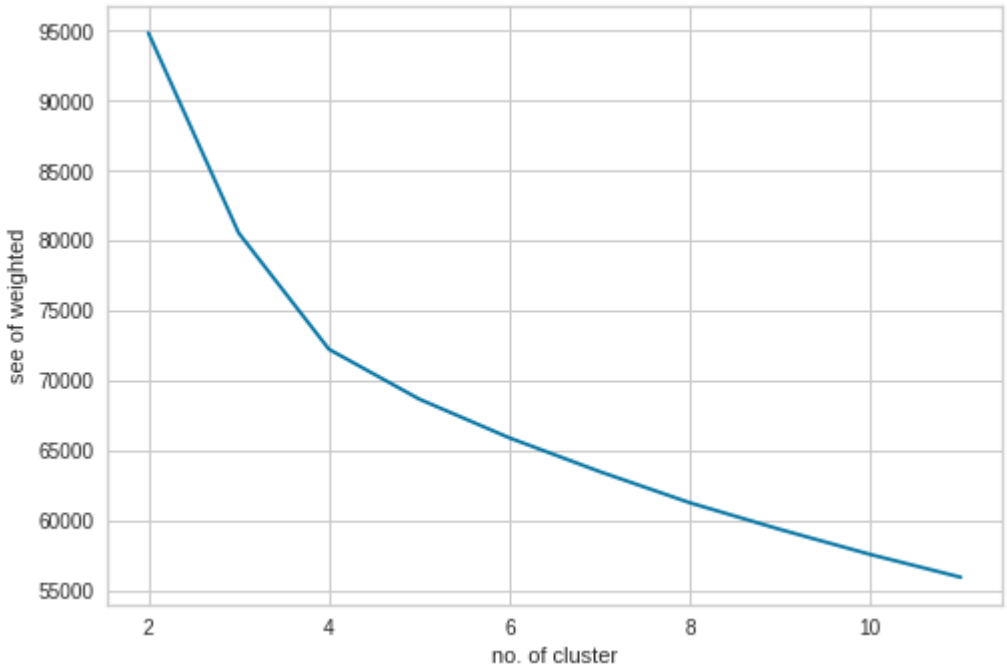


avg_w2v of KMeans of cluster 3

```
weighted_tfidf.shape
```

```
(19354, 50)
```

```
%%time
see_weighted={}
for i in range(2,12):
  k_weighted=KMeans(n_clusters=i,n_jobs=-1,max_iter=100)
  k_weighted.fit(weighted_tfidf)
  see_weighted[i]=k_weighted.inertia_
plt.plot(list(see_weighted.keys()),list(see_weighted.values()))
plt.xlabel("no. of cluster")
plt.ylabel("see of weighted")
plt.show()
```

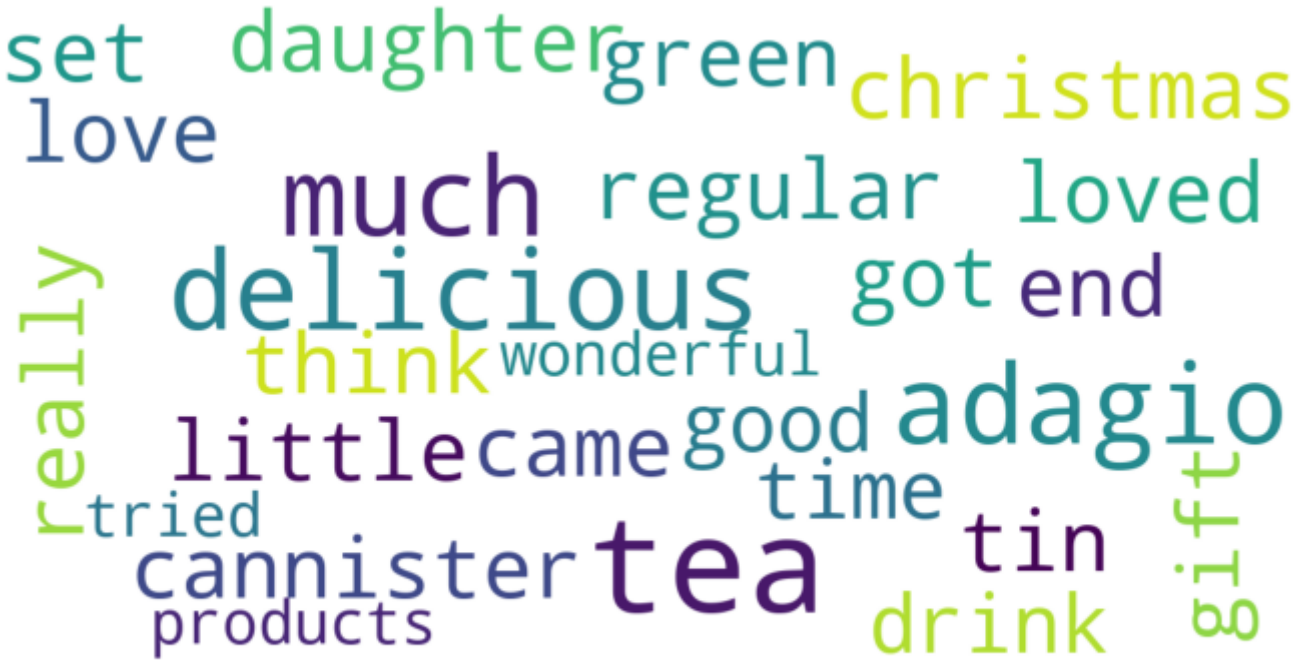



```
CPU times: user 538 ms, sys: 168 ms, total: 707 ms
Wall time: 11.1 s
```

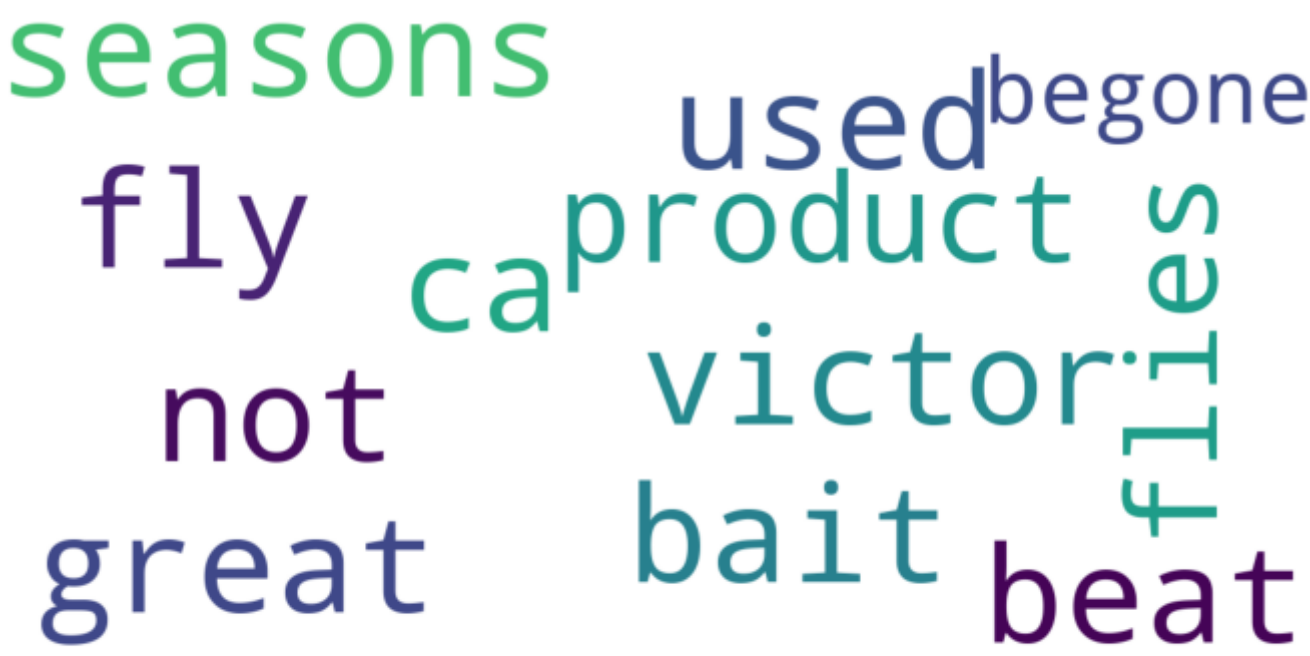
```
k_weighted=KMeans(n_clusters=4,max_iter=1000,n_jobs=-1)
k_weighted.fit(weighted_tfidf)
print(k_weighted.labels_)
print(k_weighted.cluster_centers_)
```

```
    [1 1 1 ... 0 1 1]
    [[ 1.89073147e-01  6.28611849e-04  4.44433909e-01  1.80028164e-01
       2.58635838e-01 -5.09958751e-01 -6.41803001e-01  6.68170879e-01
       3.23713147e-02  8.65370587e-01 -3.87978294e-01  5.00362726e-01
       5.70837753e-01  4.41760313e-01  4.20170943e-02 -9.45985142e-02
       7.10558583e-02  9.45014630e-01  2.17105237e-01 -2.63223219e-01
       7.90875033e-02 -2.74921412e-01 -4.70705082e-01 -7.34721855e-01
      -1.96135018e-01 -1.03585424e+00 -2.86483243e-01  8.54255152e-01
      -7.08799595e-01 -8.54005371e-02 -4.48696980e-01 -6.19707528e-01
      -2.79658471e-01  2.81177755e-01  1.54622789e-01 -3.54544807e-01
       3.74695321e-01  3.42821048e-01 -3.03169013e-01  3.20746336e-01
       4.64156192e-01 -6.54823219e-02 -5.89997850e-01  1.18985802e+00
       3.77975751e-01 -5.95220880e-01  1.33460280e-01 -1.71625251e-01
       3.77322976e-01 -7.78357434e-01]
     [ 3.30646980e-01 -7.26038428e-02  2.52934705e-01 -1.68828304e-01
      -5.21630071e-01 -2.77801983e-01 -2.71633138e-01  6.18314859e-01
       1.12557750e-01 -9.06008101e-02  1.29914212e-01  4.63517072e-01
      -1.04717865e-02  2.55925559e-01 -4.66443795e-02 -4.63935088e-02
       1.07843235e-01  2.97516726e-01  3.74748461e-02 -1.36983783e-01
      -1.41598454e-01 -4.11434286e-01  2.27083296e-01 -4.25246802e-02
      -1.10926913e-01 -4.22189297e-01 -4.62888836e-02  4.73887395e-01
      -3.44863652e-01 -7.96337267e-02 -2.46852856e-01 -2.46322184e-01
      -1.13274334e-01  3.55275980e-02 -5.03996869e-02 -2.58291619e-01
       3.75586366e-01  5.50016228e-02 -3.02853323e-01 -7.60143147e-02
      -3.82007083e-02 -2.49051314e-01 -3.54044564e-02  3.42146665e-01
       6.39270305e-01 -4.80293869e-01  5.70844616e-01  7.70512403e-02
       1.96667034e-01 -2.24025169e-01]
     [ 5.26288665e-01  9.23394358e-02 -6.64637772e-02 -3.45790160e-01
      -7.05967329e-01 -5.40480277e-01 -2.56302006e-01  6.23084676e-01
       1.62887373e-01 -6.00698671e-01  6.99416214e-01  3.93134502e-01
      -3.70572076e-01 -1.05473048e-01 -4.21848789e-01  5.03551073e-02
       3.64569923e-01  3.47400504e-01 -5.61093622e-01  5.52093575e-01
       6.54084707e-02 -5.55313850e-01  5.08886934e-01  7.08306234e-02
       8.29281788e-02 -2.25665879e-01 -4.72047918e-01  1.95979679e-01
      -5.56148046e-01  4.52694346e-02  1.28537190e-01  1.25197251e-01
       1.64738077e-01 -1.57493223e-01  3.13040285e-02 -6.01855901e-01
       5.18410331e-01 -6.60807962e-01 -1.16427702e-01 -8.35051200e-02
      -7.21191536e-01 -4.23559614e-01  3.57104189e-01  1.77139594e-01
       7.76511529e-02 -7.74841133e-01  7.07398028e-01  6.59484850e-01
       7.09111226e-01 -2.14806877e-03]
     [ 4.06633903e-01  6.16405400e-02  4.43987767e-01 -3.09896950e-02
      -1.33501303e-01 -1.00210227e-02 -4.49713465e-01  7.33489909e-01
       2.94247042e-01 -1.38002223e-01  3.14228909e-02 -9.41924205e-02
      -2.26175199e-01  1.27492773e-01  2.23802966e-02 -2.00090238e-01
       1.18835470e-01  8.45100502e-02 -1.90037363e-01 -9.96618441e-02
      -1.82416542e-01 -3.05726920e-01  4.99795968e-01  5.42430643e-02
      -9.91757077e-02 -7.64563695e-01 -9.78997675e-02  3.96433756e-01
      -4.91579580e-01  1.24348404e-01 -1.95494269e-01 -2.85878213e-01
       1.72703416e-01 -1.61573664e-01  6.69286890e-02  4.22037402e-01
       4.33552960e-01  3.28955846e-01 -7.58745027e-01  1.52655513e-01
      -1.07674468e-01 -4.07115620e-01 -2.26605666e-01  4.89823376e-01
       2.71287643e-01 -4.15510496e-01  2.20487929e-01 -1.65256377e-01
       3.32744397e-01 -2.71806513e-01]]



w_cluster0=[]
w_cluster1=[]
w_cluster2=[]
w_cluster3=[]
for i in range(k_weighted.labels_.shape[0]):
  if k_weighted.labels_[i]==0:
    w_cluster0.append(text[i])
  elif k_weighted.labels_[i]==1:
    w_cluster1.append(text[i])
```

```
  elif k_weighted.labels_[i]==2:
    w_cluster2.append(text[i])
  else:
    w_cluster3.append(text[i])


show_wordcloud(w_cluster0[0],title="weighted of KMeans of cluster 0")
show_wordcloud(w_cluster1[0],title="weighted of KMeans of cluster 1")
show_wordcloud(w_cluster2[0],title="weighted of KMeans of cluster 2")
show_wordcloud(w_cluster3[0],title="weighted of KMeans of cluster 3")
```

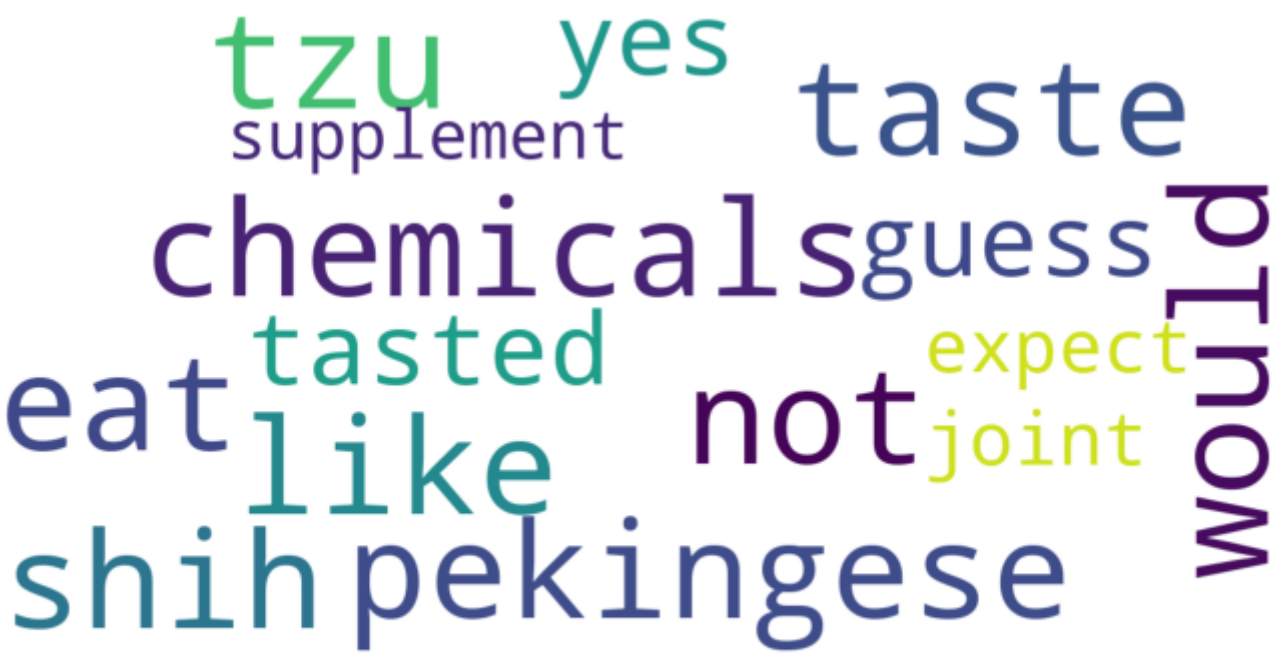


weighted of KMeans of cluster 0



weighted of KMeans of cluster 1

weighted of KMeans of cluster 2



weighted of KMeans of cluster 3

```
from prettytable import PrettyTable

pt1=PrettyTable()

pt1.add_column("KMeans with",["BOW","TFIDF","AVG-W2V","WEIGHTED"])
pt1.add_column("n_cluster",[4,4,4,4])
pt1.add_column("see: loss",[sse_bow[4],sse_tfidf[4],sse_avg_w2v[4],see_weighted[4]])
print(pt1)
```

```
+-------------+-----------+--------------------+
| KMeans with | n_cluster |     see: loss      |
+-------------+-----------+--------------------+
|     BOW     |     4     | 859945.6464110438  |
|    TFIDF    |     4     | 17941.034512925777 |
|   AVG-W2V   |     4     | 53014.80819738074  |
|   WEIGHTED  |     4     | 72224.38416493002  |
+-------------+-----------+--------------------+
```

DBSCAN

```
avg_w2v.shape
```

```
(19354, 50)
```

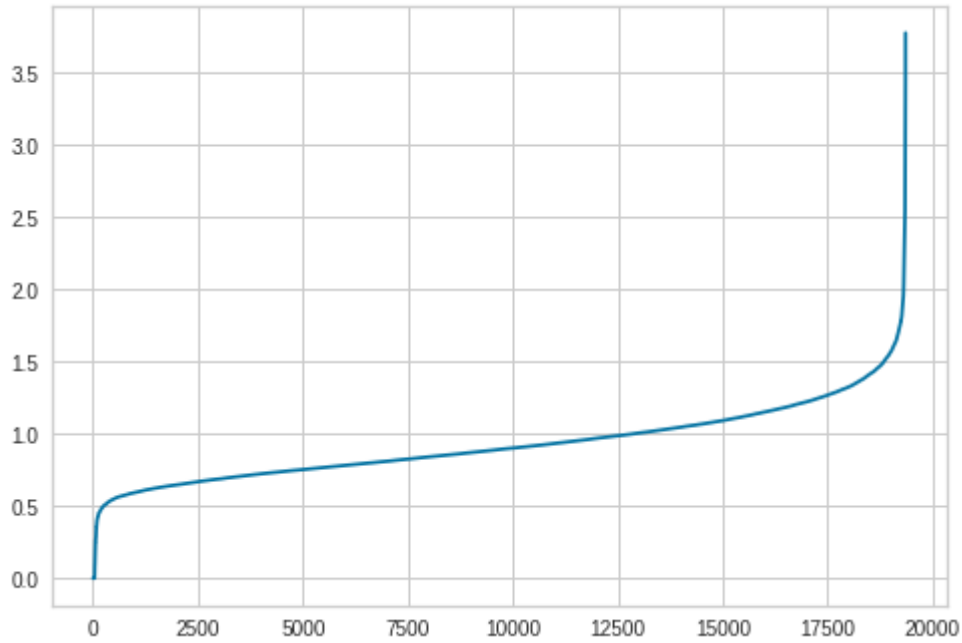
```
from scipy.cluster.hierarchy import dendrogram
from IPython.display import display
from scipy.cluster.hierarchy import dendrogram, linkage

from sklearn.cluster import DBSCAN
```

```
from sklearn.neighbors import NearestNeighbors

neigh=NearestNeighbors(n_neighbors=5,n_jobs=-1,algorithm="ball_tree")

%%time
nbrs=neigh.fit(avg_w2v)
distance,neighbors=nbrs.kneighbors(avg_w2v)
print(distance,neighbors)
```

```
[[0.         0.92980096 0.94722994 0.96110726 0.96770157]
 [0.         0.60581331 0.61103037 0.67558239 0.71845633]
 [0.         0.79192235 0.80175396 0.92739204 0.93265579]
 ...
 [0.         0.71590457 0.7505519  0.76929514 0.78351304]
 [0.         0.82177853 0.89391057 0.9452086  0.96336627]
 [0.         1.17929438 1.25906922 1.27718723 1.2806533 ]] [[    0 10368  4359 13995  6
 [    1  5206  8622  8601  8617]
 [    2  6830  4863  4443  5648]
 ...
 [19351 12626  8746  8451 12573]
 [19352 12631 15235 12554 17049]
 [19353 13992 13616  2163 15659]]
CPU times: user 50.2 s, sys: 0 ns, total: 50.2 s
Wall time: 12.8 s
```

```
distances = np.sort(distance, axis=0)
distances = distances[:,1]
plt.plot(distances)
```

```
[<matplotlib.lines.Line2D at 0x7f7e44981a20>]
```




```
%%time
dbscan_label=[]
a_eps=[0.6,1.0,1.25,1.5]
for i in a_eps:
  avg_dbscan=DBSCAN(eps=i,min_samples=100,algorithm="ball_tree",n_jobs=-1)
  avg_dbscan.fit(avg_w2v)
  dbscan_label.append(avg_dbscan.labels_)
```

```
    CPU times: user 2min 49s, sys: 0 ns, total: 2min 49s
    Wall time: 43.8 s
```

```
avg_dbscan=DBSCAN(eps=1.5,min_samples=100,algorithm="ball_tree",n_jobs=-1)
avg_dbscan.fit(avg_w2v)
```

```
DBSCAN(algorithm='ball_tree', eps=1.5, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=100, n_jobs=-1, p=None)
```

```
avg_dbscan.core_sample_indices_
```

```
array([    0,     1,     2, ..., 19350, 19351, 19352])
```

```
avg_dbscan.components_
```

```
array([[ 0.22309204,  0.025522  ,  0.45444356, ...,  0.09219588,
          0.02189925, -0.29063164],
       [ 0.47320673, -0.0767499 ,  0.25637005, ..., -0.04364521,
          0.10830297, -0.27697879],
       [ 0.40810907,  0.20621114,  0.48317144, ...,  0.18046093,
          0.04411893, -0.21277093],
       ...,
       [ 0.31749914,  0.04407222,  0.38469573, ..., -0.21330246,
          0.18817721, -0.32663914],
       [ 0.28057432, -0.01824763,  0.4180526 , ..., -0.27279043,
          0.08662378, -0.46799384],
       [ 0.37774316,  0.04203214,  0.25056939, ..., -0.17631286,
          0.04192664, -0.5340458 ]])
```

```
print(avg_dbscan.labels_)
```

```
[0 0 0 ... 0 0 0]
```

```
import collections
```

```
collections.Counter(avg_dbscan.labels_)
```

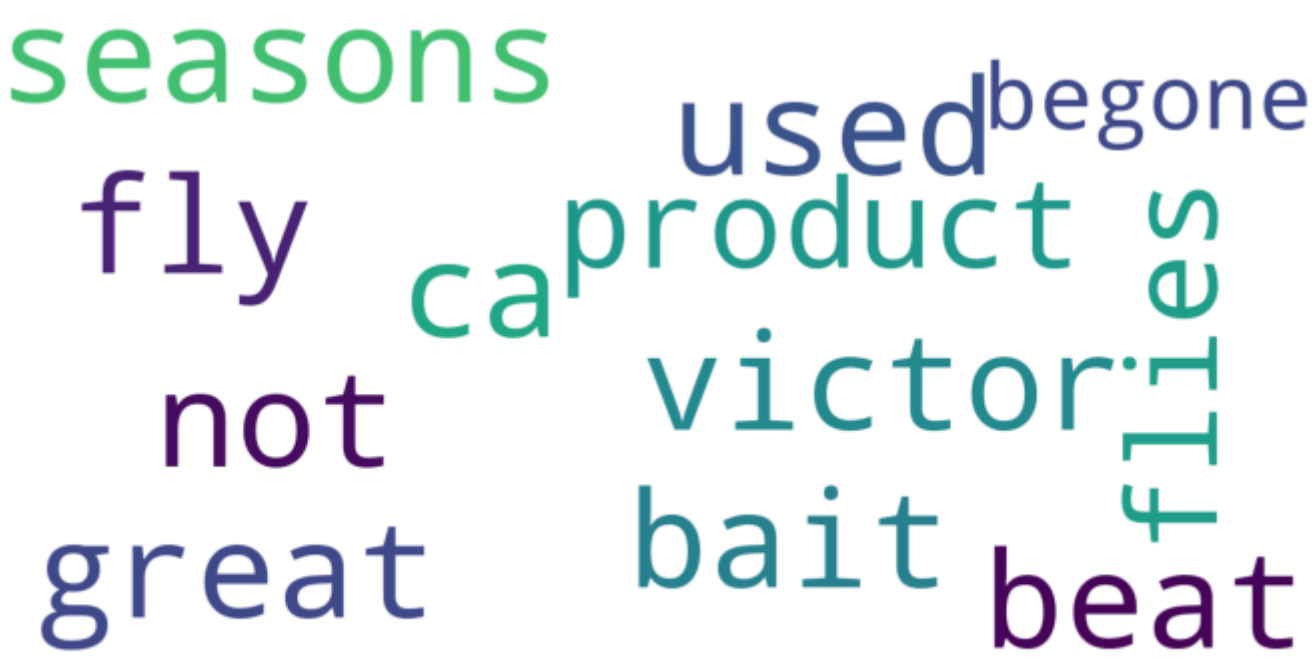
```
Counter({-1: 1110, 0: 18244})
```

```
final.Score.value_counts()
```

```
1    16339
0     3015
Name: Score, dtype: int64
```

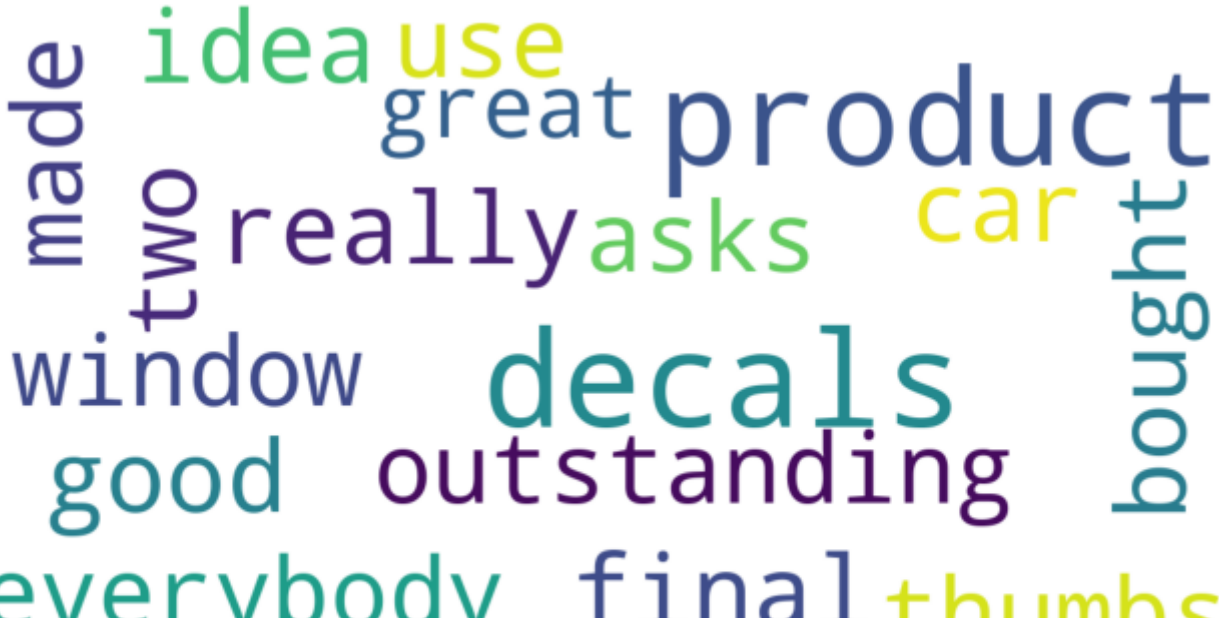
```
db_cluster0=[]
db_cluster1=[]
for i in range(avg_dbscan.labels_.shape[0]):
  if avg_dbscan.labels_[i]==0:
    db_cluster0.append(text[i])
  else:
    db_cluster1.append(text[i])
```

```
for i in range(10):
  show_wordcloud(db_cluster0[i],"dbscan on cluster"+str(i)+ "zero")
```

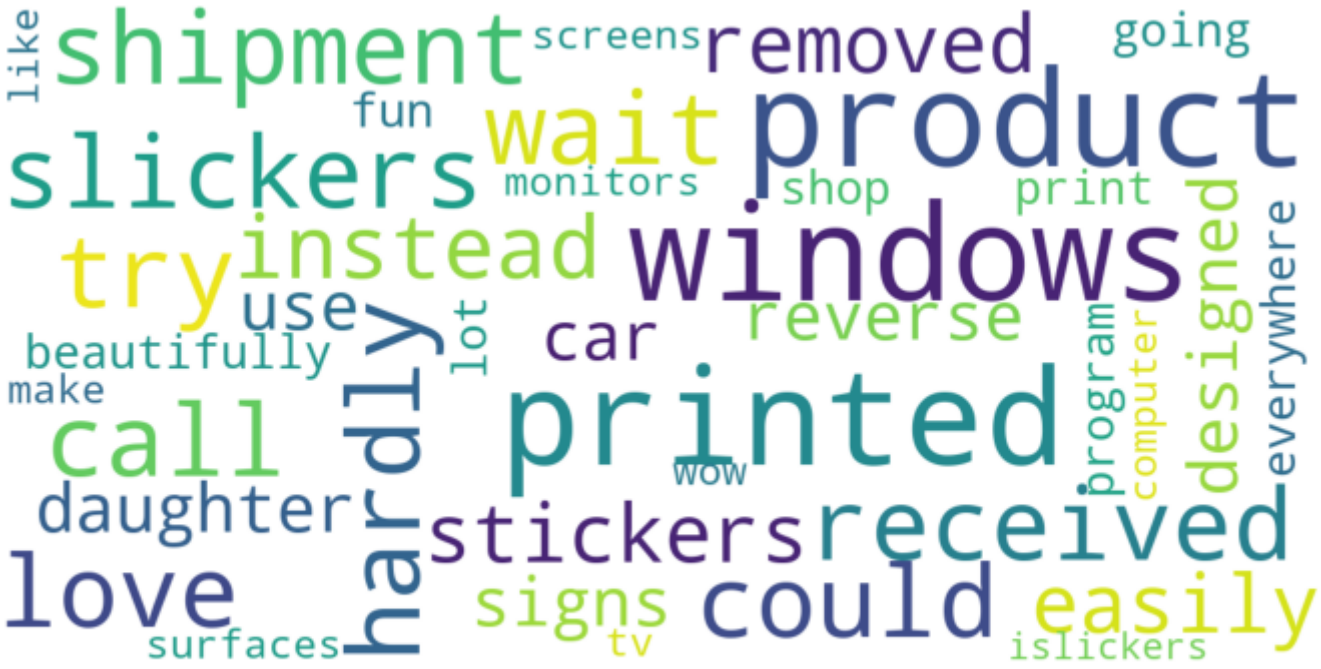


dbscan on cluster0zero

dbscan on cluster1zero

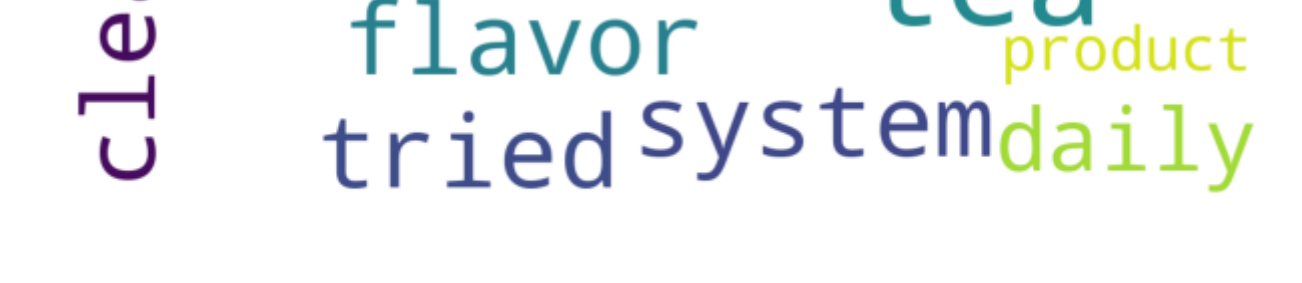


dbscan on cluster2zero

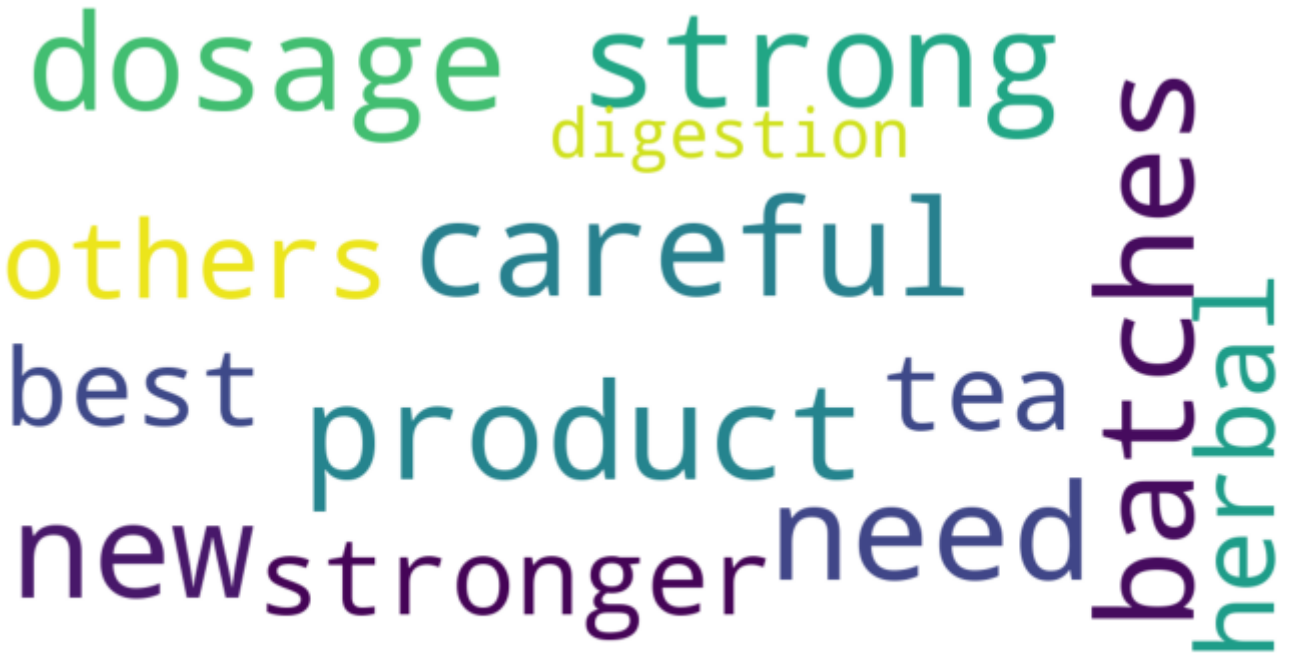


dbscan on cluster3zero

dbscan on cluster4zero



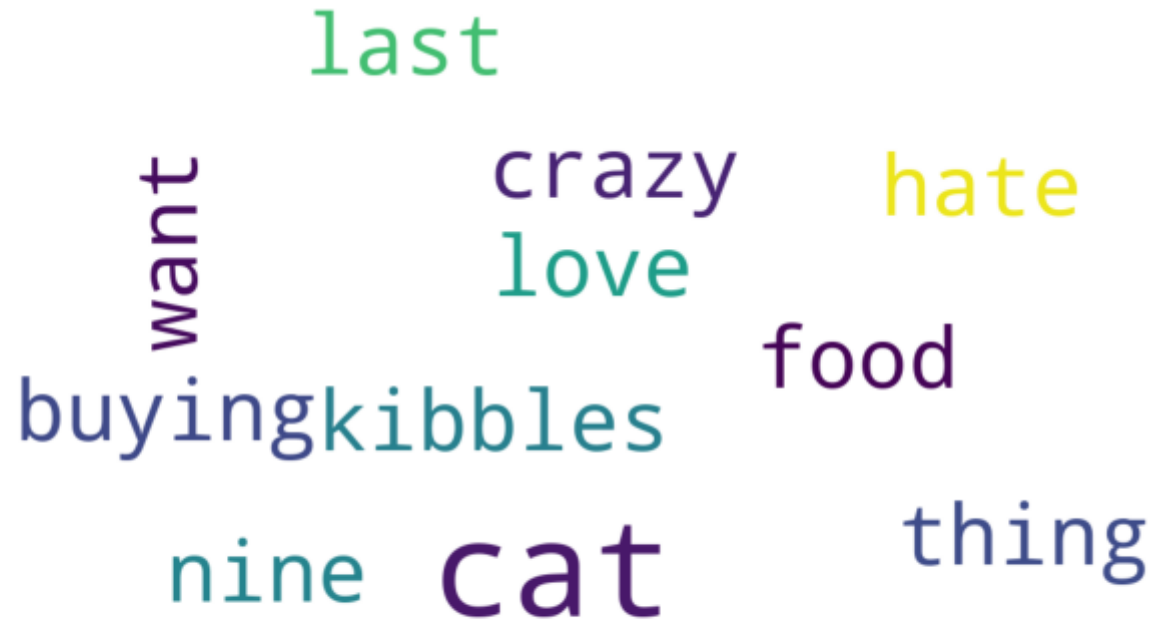
dbscan on cluster5zero



dbscan on cluster6zero

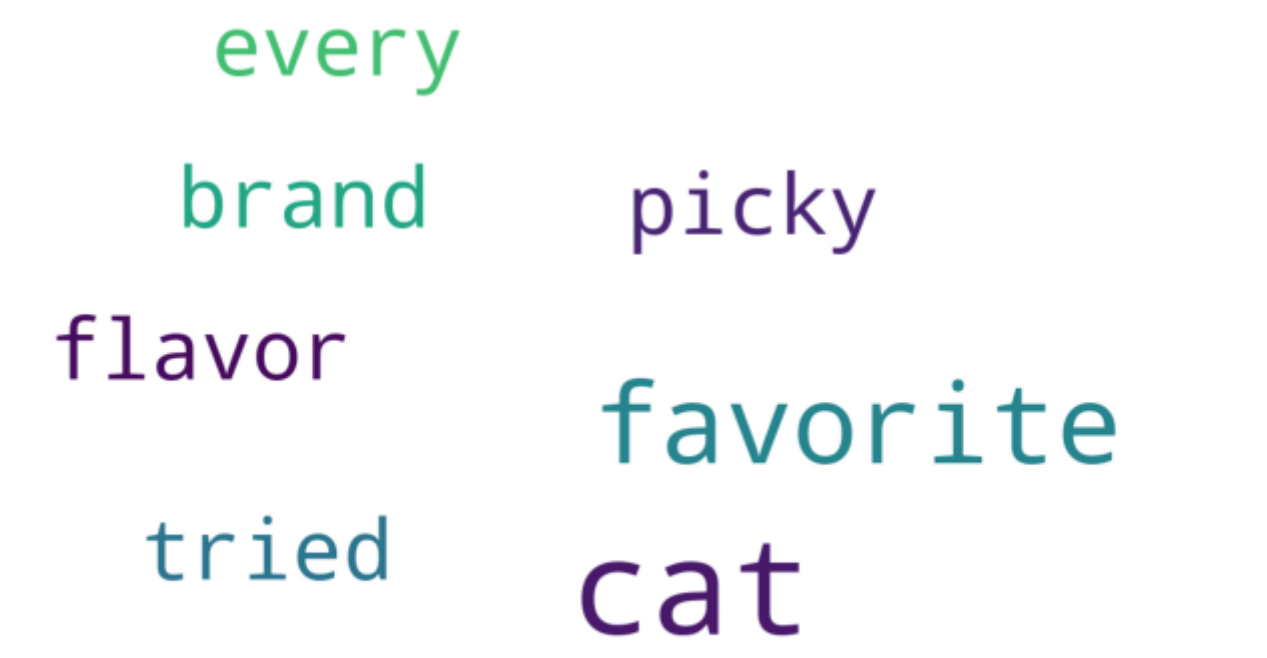
dbscan on cluster7zero

dbscan on cluster8zero

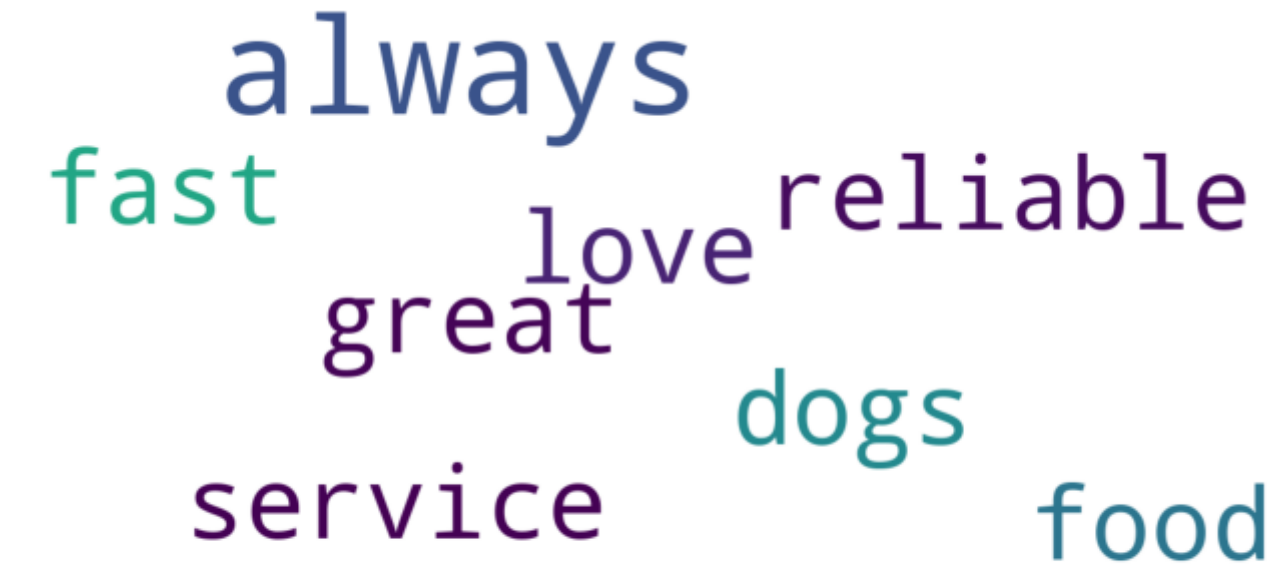


dbscan on cluster9zero

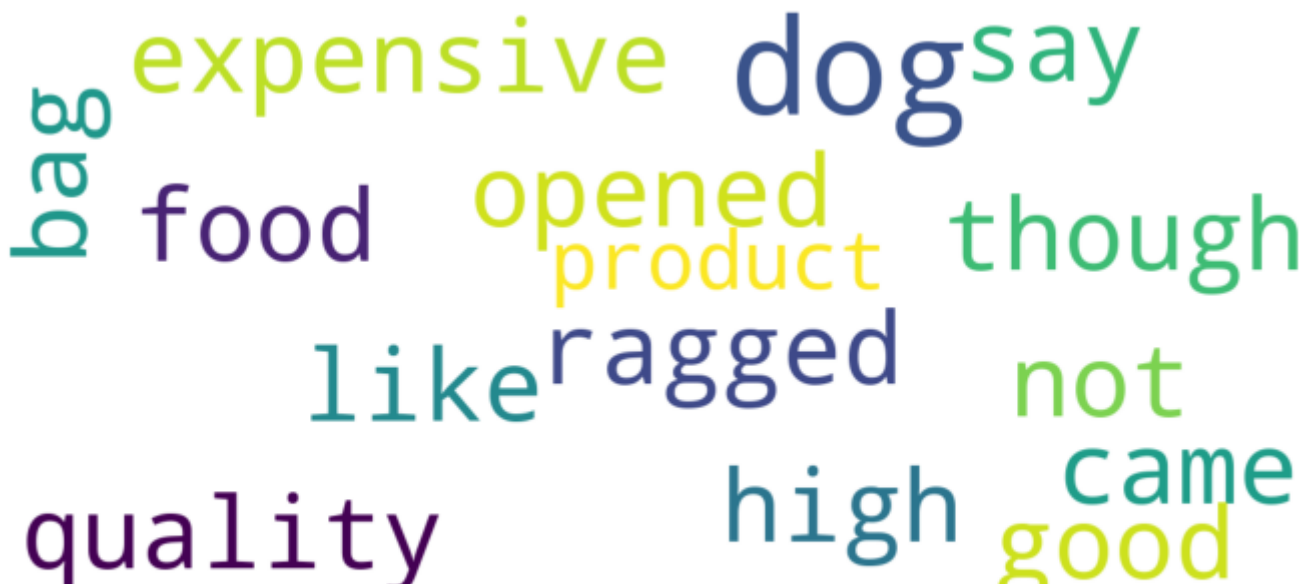
```
for i in range(10):
  show_wordcloud(db_cluster1[i],"dbscan on cluster"+str(i)+ "negative")
```



dbscan on cluster0negative

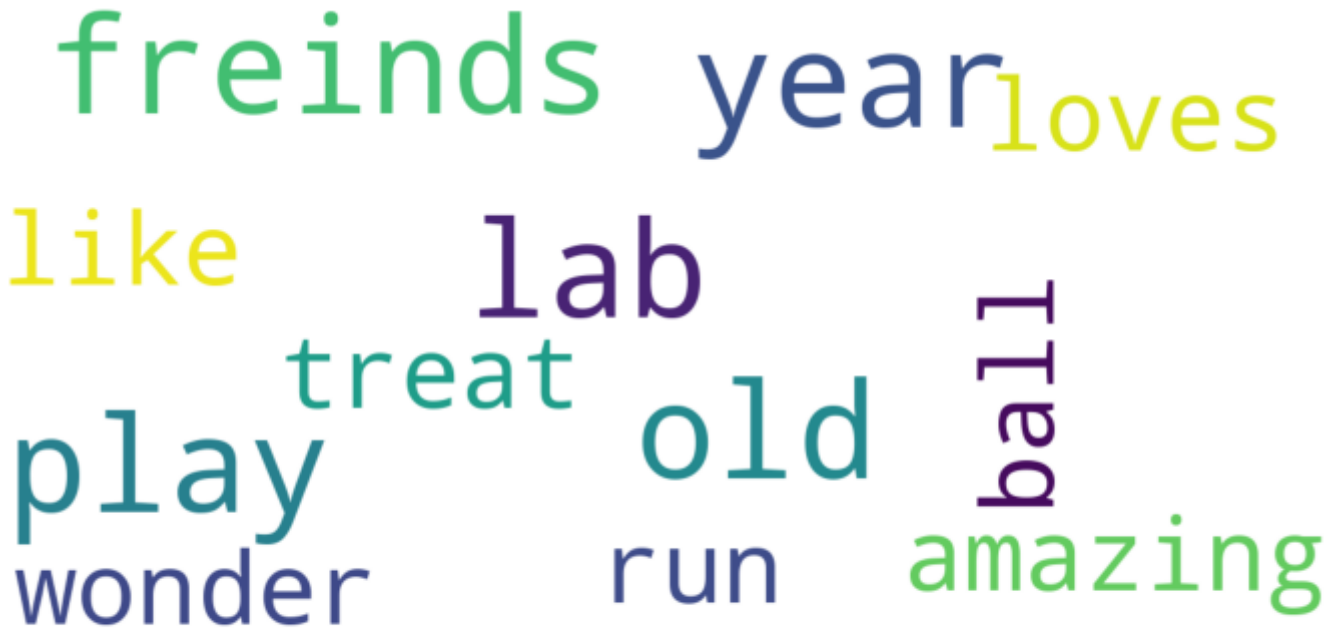


dbscan on cluster1negative



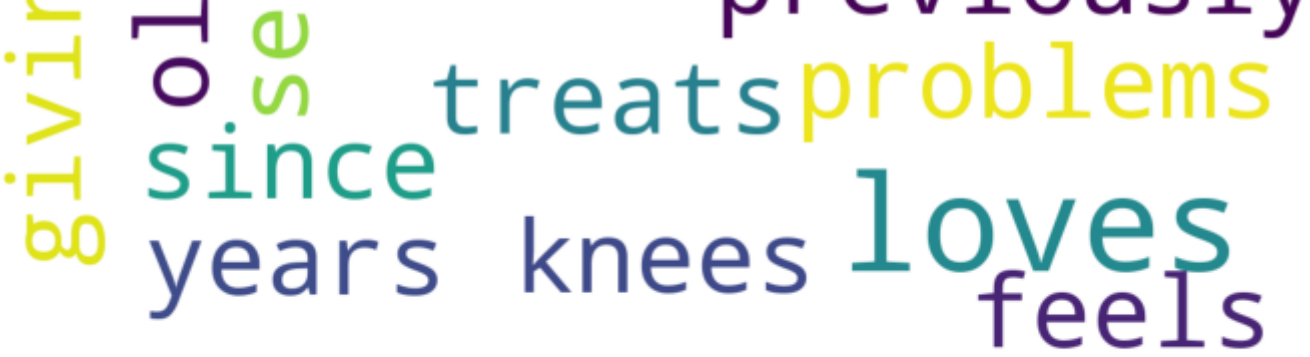
dbscan on cluster2negative

dbscan on cluster3negative

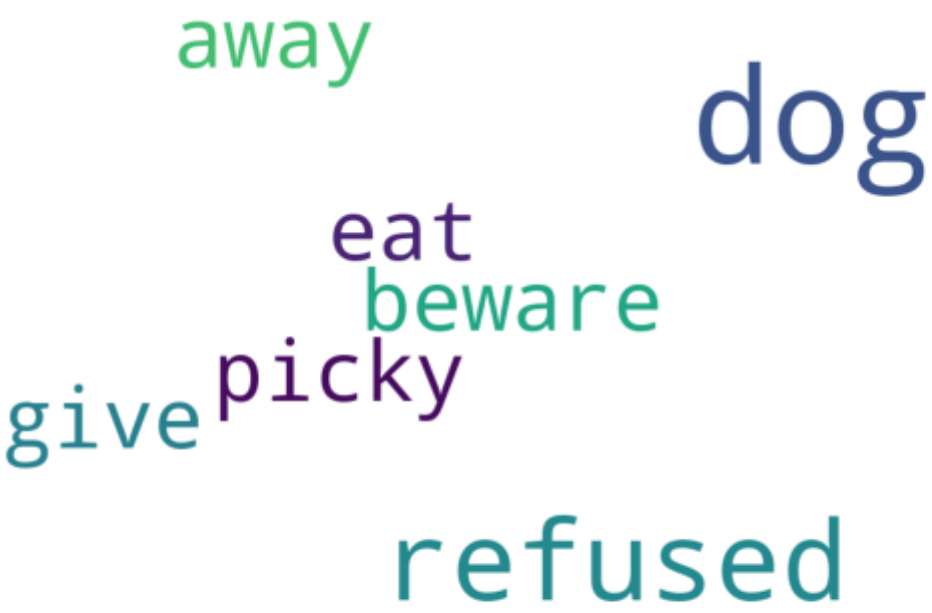


dbscan on cluster4negative


best
loves
dbscan on cluster5negative
treats
supposed
dog
give
healthy
continue
love
try
good
goldie
likes
dbscan on cluster6negative
give
free
great
love
make
happy
price
dog
get
zukes
treat
shipping
dbscan on cluster7negative
picky
dog
better
really
previously



dbscan on cluster8negative



dbscan on cluster9negative

```
avg_dbscan_1=DBSCAN(eps=1.25,min_samples=100,algorithm="ball_tree",n_jobs=-1)
avg_dbscan_1.fit(avg_w2v)
```

```
DBSCAN(algorithm='ball_tree', eps=1.25, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=100, n_jobs=-1, p=None)
```

```
dbscan_label[0]
```

```
array([-1, -1, -1, ..., -1, -1, -1])
```

```
collections.Counter(dbscan_label[0])
```

```
Counter({-1: 19354})
```

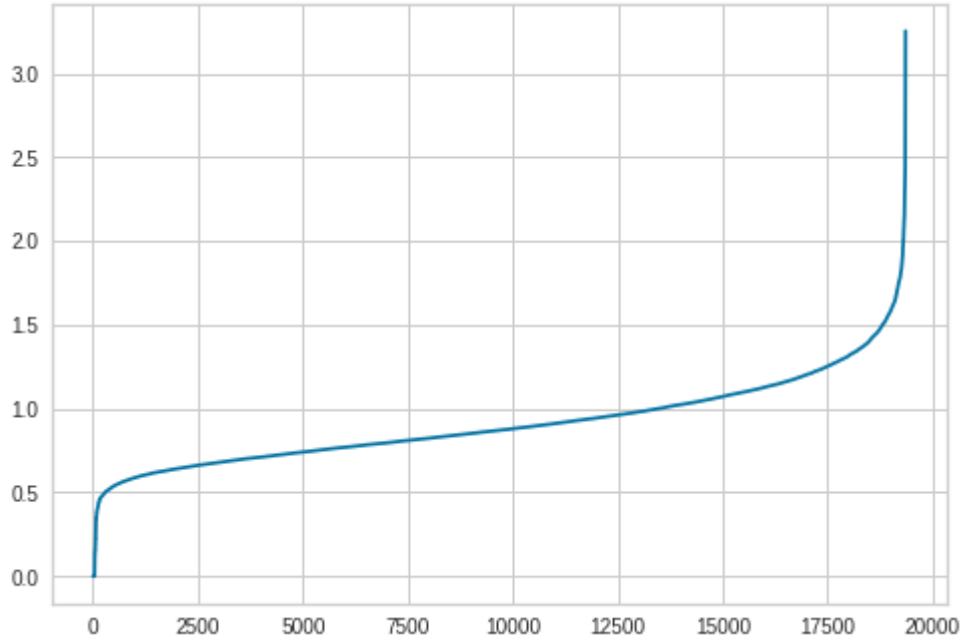
```
collections.Counter(dbscan_label[1])
```

```
Counter({-1: 11299, 0: 8055})
```

```
collections.Counter(dbscan_label[2])
```

```
%%time
nbrs=neigh.fit(weighted_tfidf)
distance,neighbors=nbrs.kneighbors(weighted_tfidf)
print(distance,neighbors)
distances = np.sort(distance, axis=0)
distances = distances[:,1]
plt.plot(distances)
```

```
[[0.         0.65178084 0.70014927 0.74172605 0.76100029]
 [0.         0.47914398 0.51811005 0.565888   0.5875039 ]
 [0.         0.74785159 0.77860093 0.78113972 0.79635951]
 ...
 [0.         0.70319514 0.71673645 0.74730913 0.75560323]
 [0.         0.9146373  0.95872628 0.9600704  0.97467584]
 [0.         1.01610541 1.08163867 1.08270919 1.0890117 ]] [[    0  2576 10368  3036  9
 [    1  5206 10484  8622  8572]
 [    2 10368  8580  4443     3]
 ...
 [19351  8451 15547  8958 15018]
 [19352  3309  7738 17812 14745]
 [19353 10733  6422 13616 15603]]
CPU times: user 43.2 s, sys: 0 ns, total: 43.2 s
Wall time: 11 s
```




```
%%time
dbscan_label={}
a_eps=[0.5,1.3,1.4,1.5]
for i in a_eps:
  w_dbscan=DBSCAN(eps=i,min_samples=100,algorithm="ball_tree",n_jobs=-1)
  w_dbscan.fit(weighted_tfidf)
  dbscan_label[i]=w_dbscan.labels_
```

```
CPU times: user 2min 26s, sys: 0 ns, total: 2min 26s
Wall time: 37.9 s
```

```
collections.Counter(dbscan_label[0.5])
```

```
Counter({-1: 19354})
```

```
collections.Counter(dbscan_label[1.3])
```

```
Counter({-1: 4588, 0: 14766})
```

```
collections.Counter(dbscan_label[1.4])
```

```
Counter({-1: 2959, 0: 16395})
```

```
collections.Counter(dbscan_label[1.5])
```

```
Counter({-1: 1880, 0: 17474})
```

```
w_dbscan=DBSCAN(eps=1.5,min_samples=100,algorithm="ball_tree",n_jobs=-1)
w_dbscan.fit(weighted_tfidf)
```

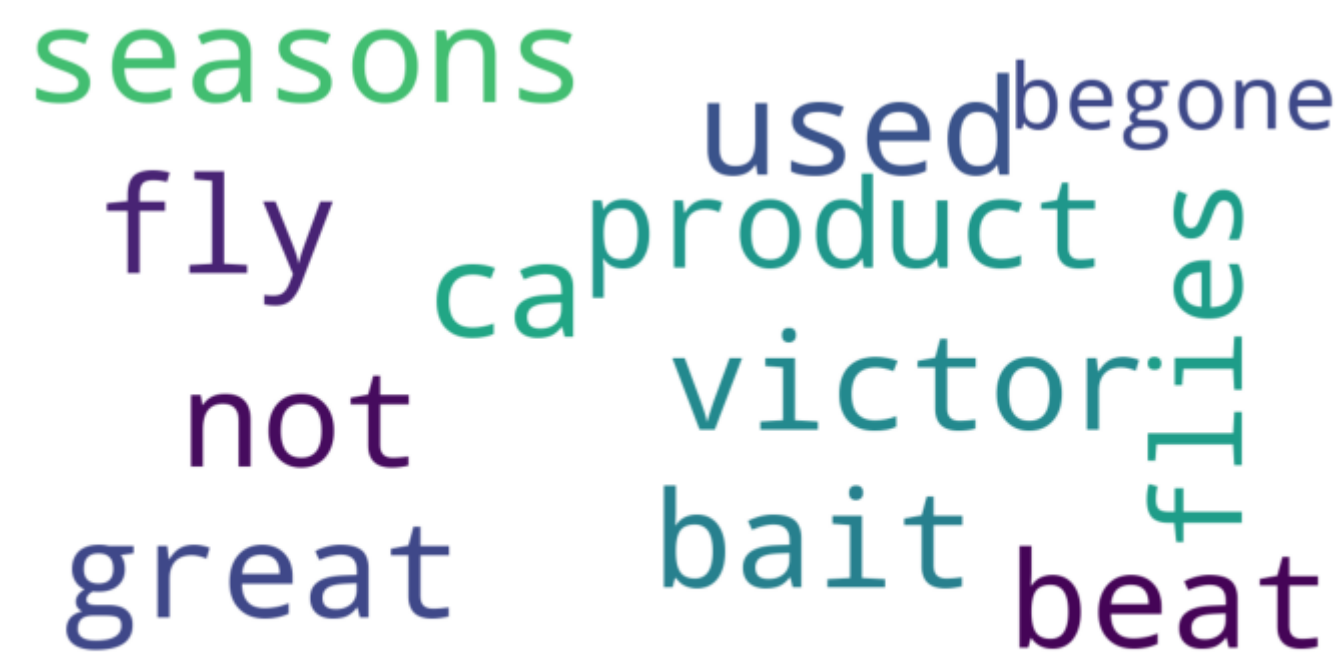
```
DBSCAN(algorithm='ball_tree', eps=1.5, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=100, n_jobs=-1, p=None)
```

```
db_w_cluster0=[]
db_w_cluster1=[]
for i in range(w_dbscan.labels_.shape[0]):
  if w_dbscan.labels_[i]==0:
    db_w_cluster0.append(text[i])
  else:
    db_w_cluster1.append(text[i])
```

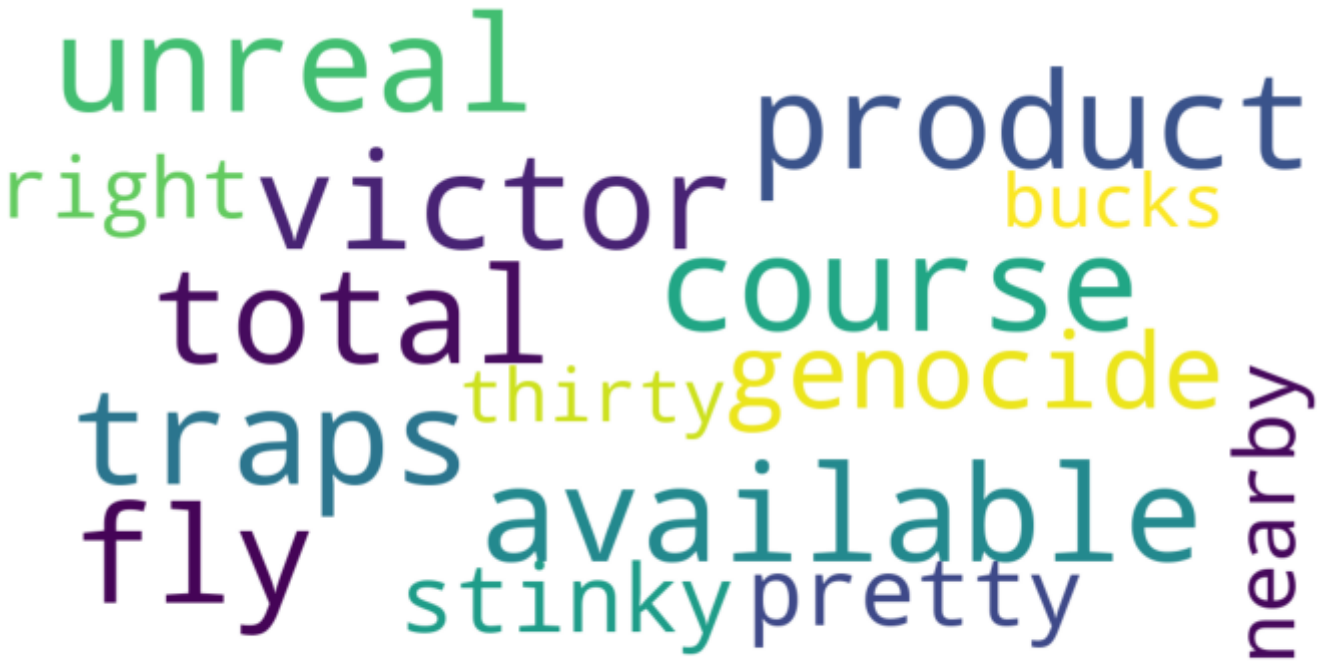
```
len(db_w_cluster0)
```

```
17474
```

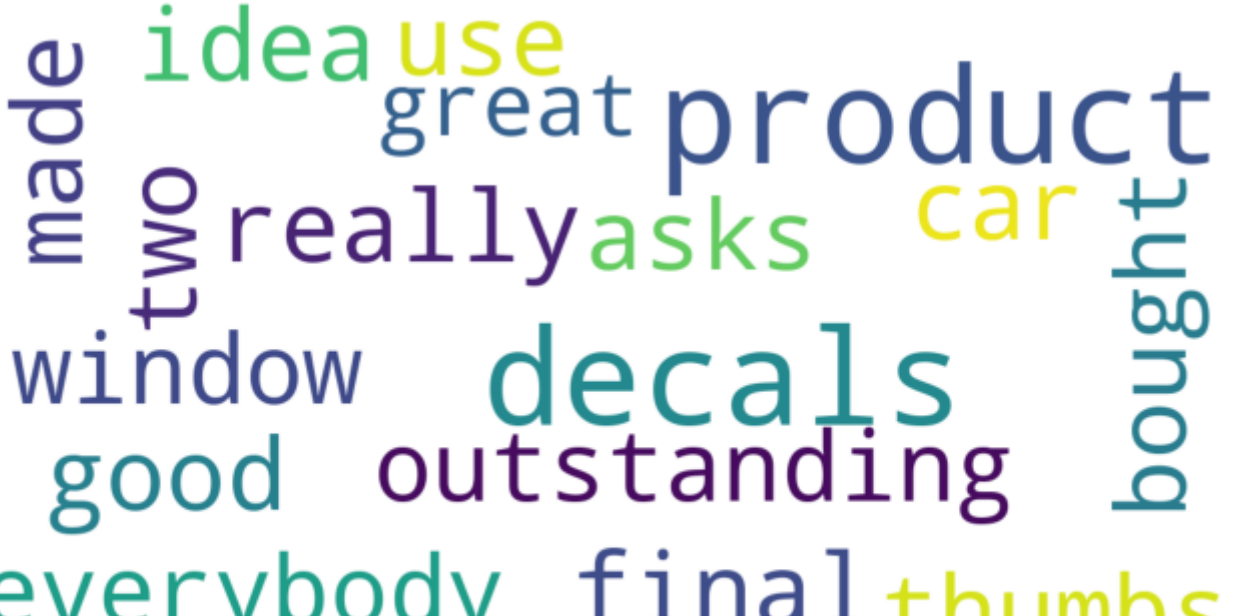
```
for i in range(3):
  show_wordcloud(db_cluster0[i],"dbscan on cluster"+str(i)+ "zero")
for i in range(3):
  show_wordcloud(db_cluster1[i],"dbscan on cluster"+str(i)+ "outliers")
```



dbscan on cluster0zero



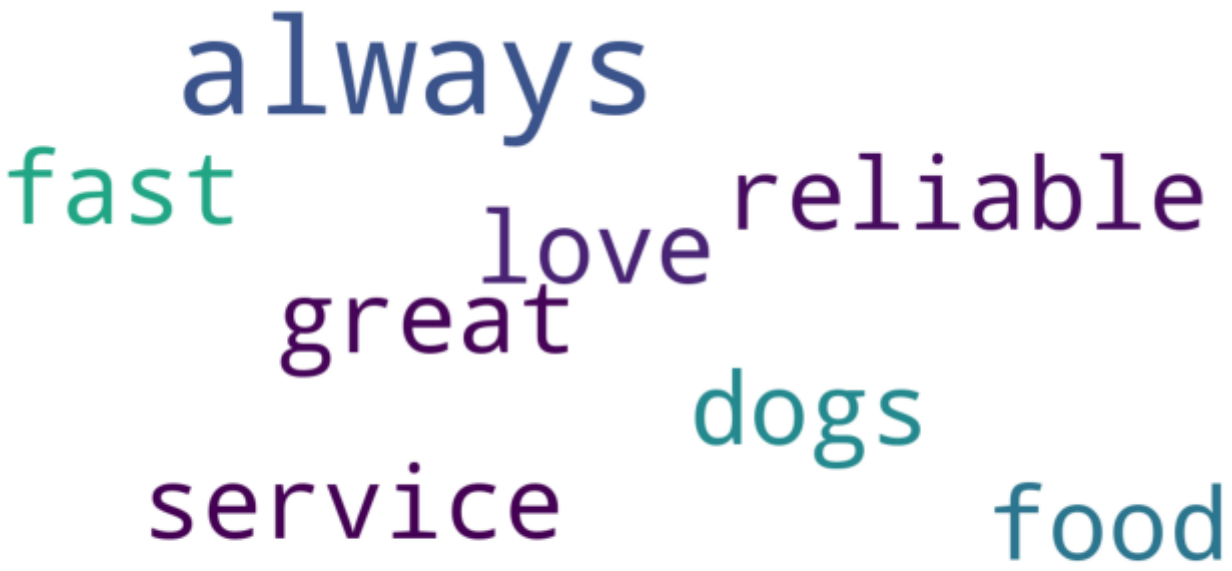
dbscan on cluster1zero



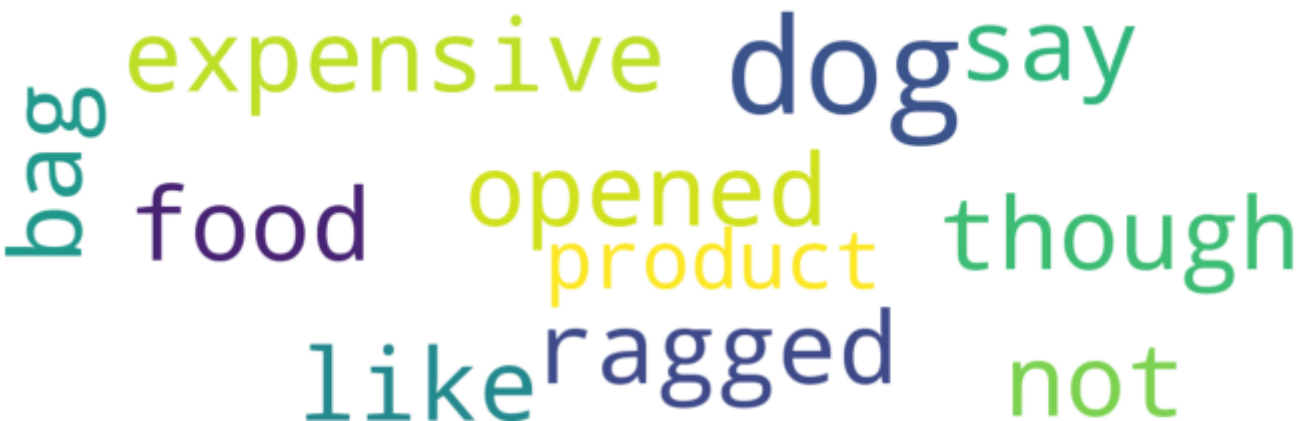
everybody thumbs

dbscan on cluster2zero

dbscan on cluster0outliers



dbscan on cluster1outliers



dbscan on cluster2outliers

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

```
%%time
a_avg=AgglomerativeClustering(n_clusters=3,linkage="complete")
a_avg.fit(avg_w2v)
```

```
CPU times: user 23.9 s, sys: 39.5 ms, total: 24 s
Wall time: 23.9 s
```

```
a_avg.labels_
```

```
array([2, 2, 0, ..., 2, 2, 0])
```

```
a_avg.children_
```

```
array([[  251,  4827],
       [ 9878,  9886],
       [11583, 11584],
       ...,
       [38684, 38702],
       [38703, 38704],
       [38248, 38705]])
```

```
ag_a3_cluster0=[]
ag_a3_cluster1=[]
ag_a3_cluster2=[]

for i in range(a_avg.labels_.shape[0]):
  if a_avg.labels_[i]==0:
    ag_a3_cluster0.append(text[i])
  elif a_avg.labels_[i]==1:
    ag_a3_cluster1.append(text[i])
  else:
    ag_a3_cluster2.append(text[i])
```

```
collections.Counter(a_avg.labels_)
```

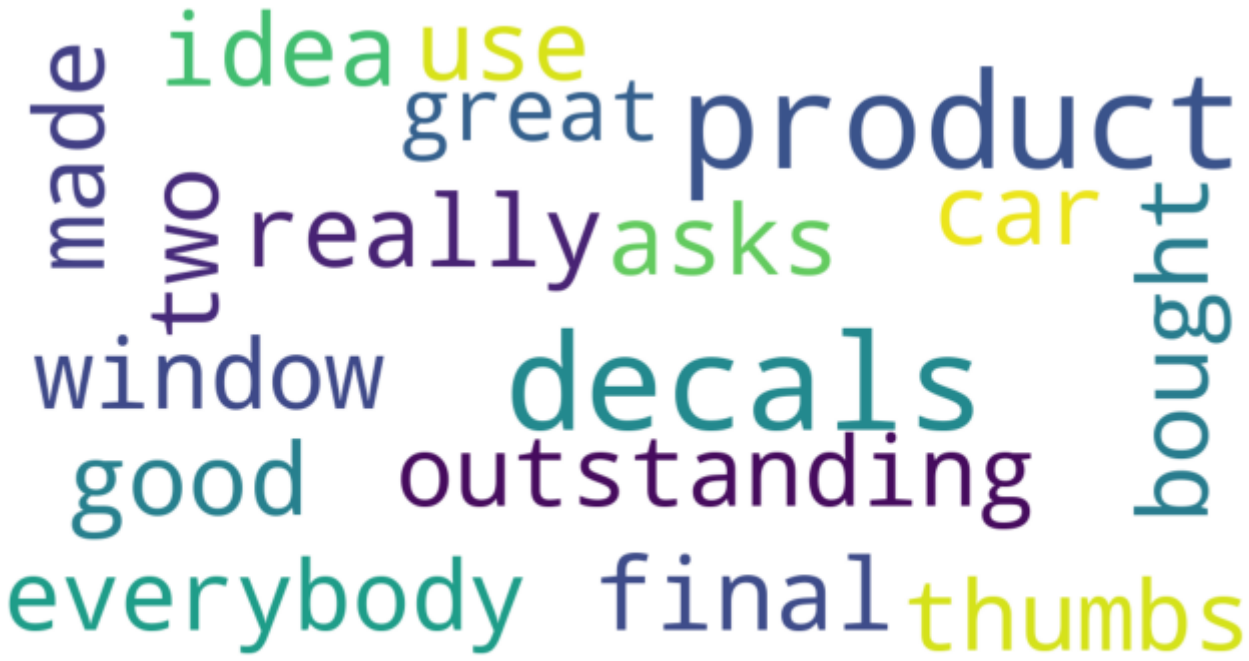
```
Counter({0: 13433, 1: 2, 2: 5919})
```

```
len(ag_a3_cluster1)
```

```
2
```

```
for i in range(2):
  show_wordcloud(ag_a3_cluster0[i],"cluster 0")
for i in range(2):
```

```
  show_wordcloud(ag_a3_cluster1[i],"cluster 1")
for i in range(2):
  show_wordcloud(ag_a3_cluster2[i],"cluster 2")
```

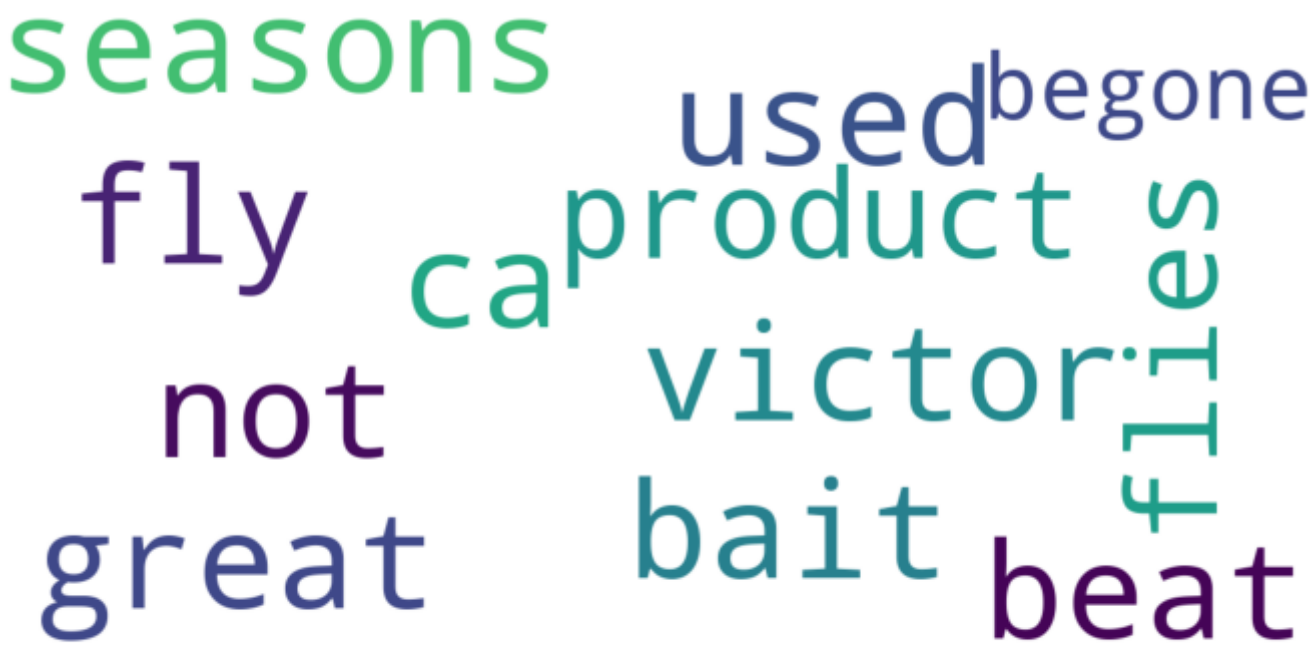


cluster 0

cluster 0

cluster 1

cluster 1



cluster 2

cluster 2

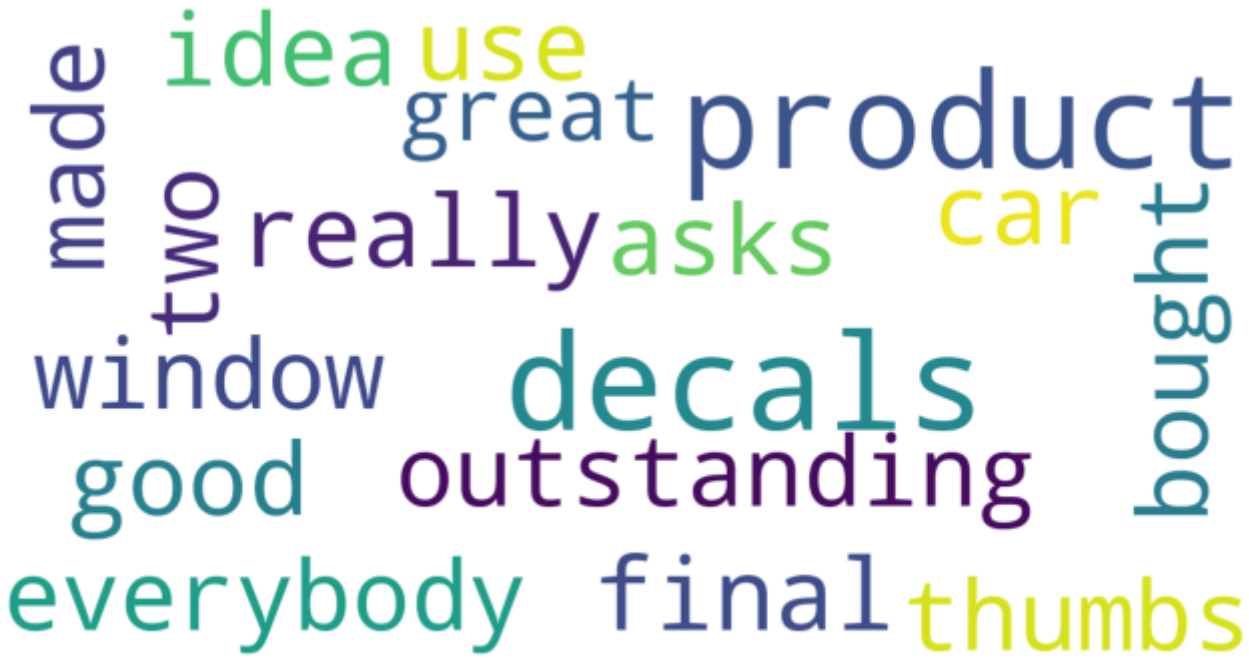
```
%%time
a_avg_5=AgglomerativeClustering(n_clusters=5,linkage="complete")
a_avg_5.fit(avg_w2v)
```

```
CPU times: user 23.9 s, sys: 38 ms, total: 23.9 s
Wall time: 23.9 s
```

```
collections.Counter(a_avg_5.labels_)
```

```
Counter({0: 11674, 1: 3021, 2: 2898, 3: 2, 4: 1759})
```

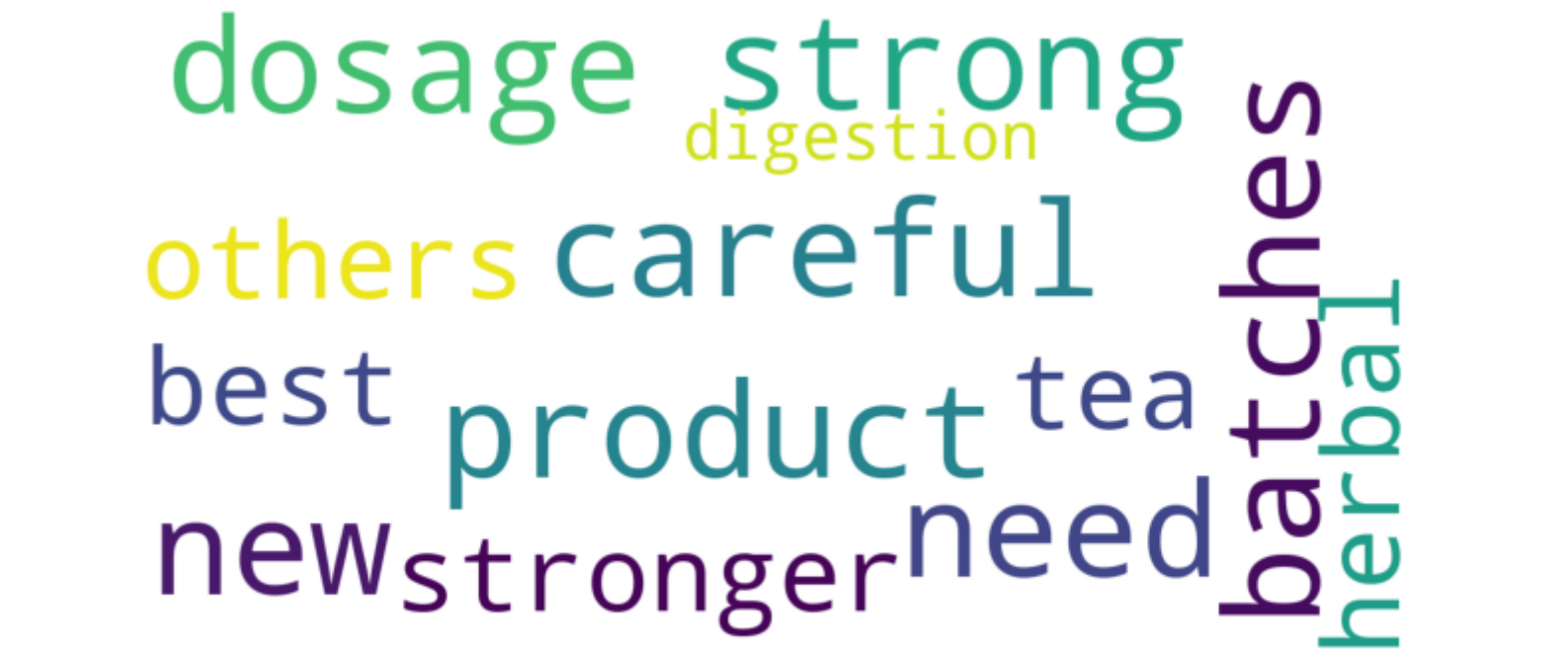
```
ag_a5_cluster0=[]
ag_a5_cluster1=[]
ag_a5_cluster2=[]
ag_a5_cluster3=[]
ag_a5_cluster4=[]

for i in range(a_avg_5.labels_.shape[0]):
  if a_avg_5.labels_[i]==0:
    ag_a5_cluster0.append(text[i])
  elif a_avg_5.labels_[i]==1:
    ag_a5_cluster1.append(text[i])
  elif a_avg_5.labels_[i]==2:
    ag_a5_cluster2.append(text[i])
  elif a_avg_5.labels_[i]==3:
    ag_a5_cluster3.append(text[i])
  else:
    ag_a5_cluster4.append(text[i])


for i in range(2):
  show_wordcloud(ag_a5_cluster0[i],"cluster 0")
for i in range(2):
  show_wordcloud(ag_a5_cluster1[i],"cluster 1")
for i in range(2):
  show_wordcloud(ag_a5_cluster2[i],"cluster 2")
for i in range(2):
  show_wordcloud(ag_a5_cluster3[i],"cluster 3")
for i in range(2):
  show_wordcloud(ag_a5_cluster4[i],"cluster 4")
```



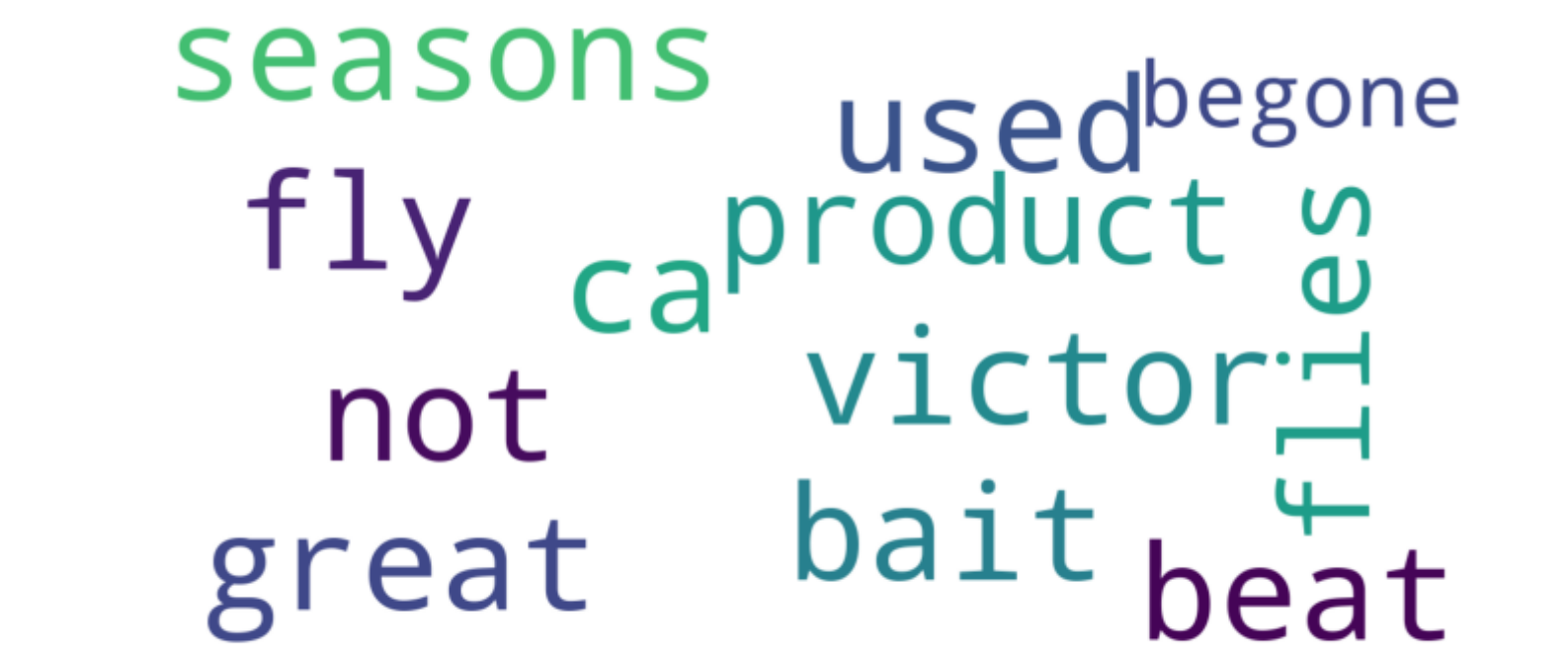
cluster 0

cluster 0

cluster 1



cluster 1



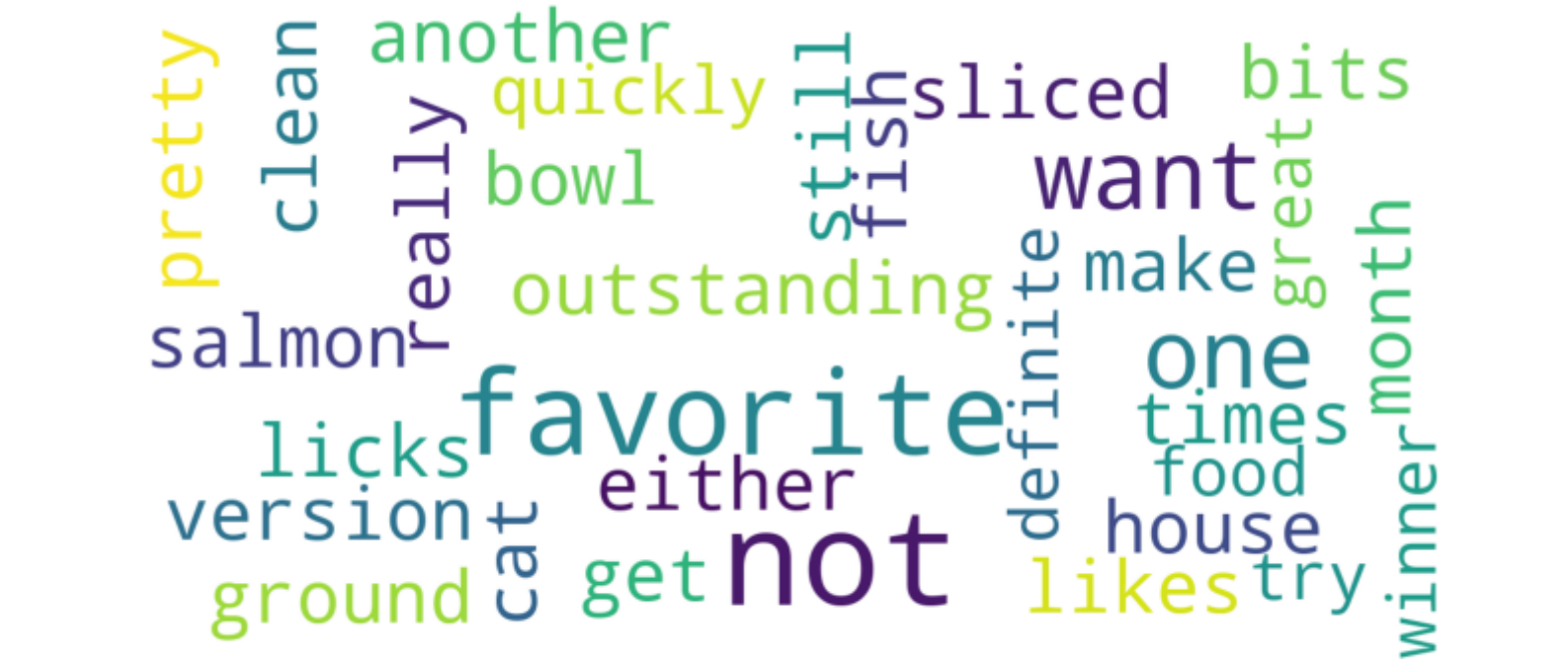
cluster 2

cluster 2

cluster 3

cluster 3

cluster 4



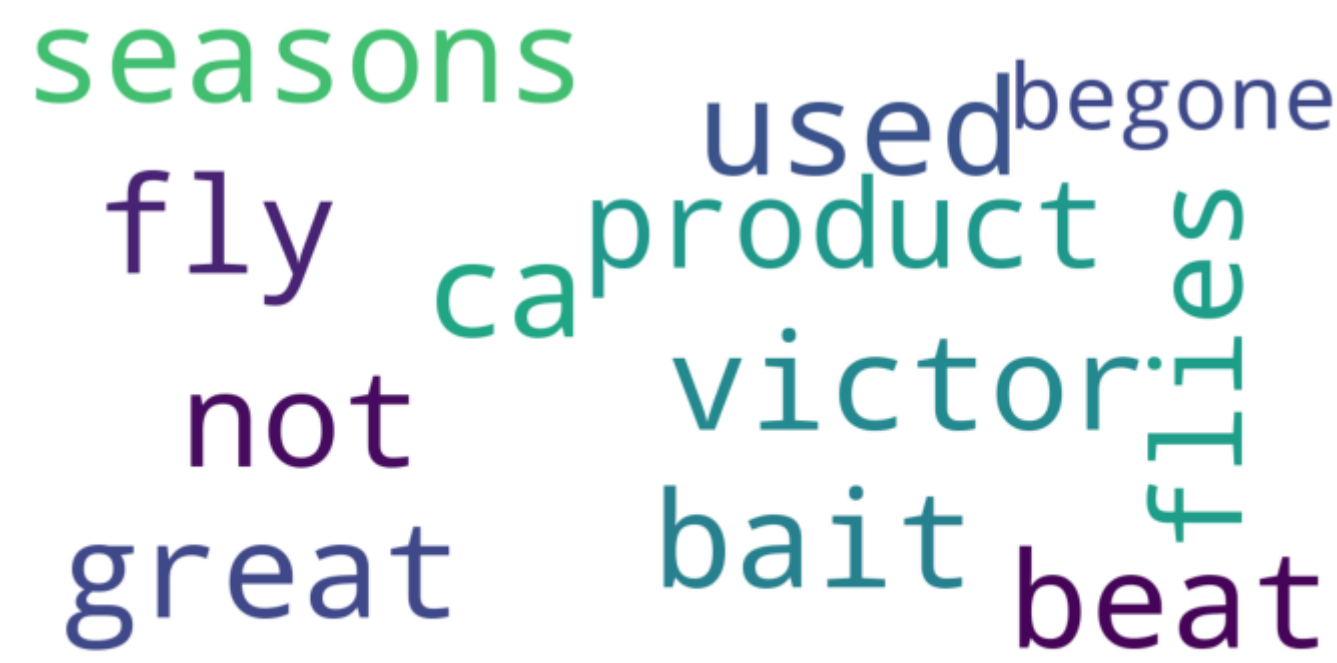
cluster 4

```
%%time
w_agg=AgglomerativeClustering(n_clusters=5,linkage="complete")
w_agg.fit(weighted_tfidf)
```

```
CPU times: user 23.4 s, sys: 27.8 ms, total: 23.4 s
Wall time: 23.4 s
```

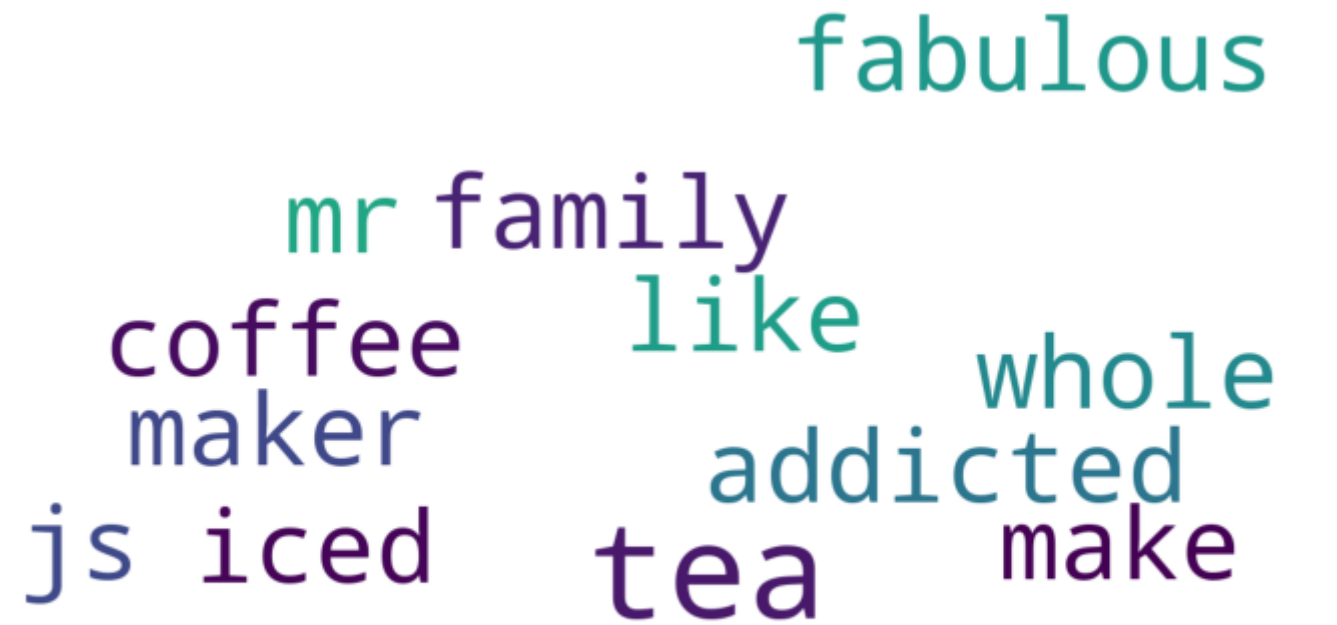
```
w_a5_cluster0=[]
w_a5_cluster1=[]
w_a5_cluster2=[]
w_a5_cluster3=[]
w_a5_cluster4=[]

for i in range(w_agg.labels_.shape[0]):
  if w_agg.labels_[i]==0:
    w_a5_cluster0.append(text[i])
  elif w_agg.labels_[i]==1:
    w_a5_cluster1.append(text[i])
  elif w_agg.labels_[i]==2:
    w_a5_cluster2.append(text[i])
  elif w_agg.labels_[i]==3:
    w_a5_cluster3.append(text[i])
```

```
  else:
    w_a5_cluster4.append(text[i])


for i in range(2):
  show_wordcloud(w_a5_cluster0[i],"cluster 0")
for i in range(2):
  show_wordcloud(w_a5_cluster1[i],"cluster 1")
for i in range(2):
  show_wordcloud(w_a5_cluster2[i],"cluster 2")
for i in range(2):
  show_wordcloud(w_a5_cluster3[i],"cluster 3")
for i in range(2):
  show_wordcloud(w_a5_cluster4[i],"cluster 4")
```



cluster 0

cluster 0

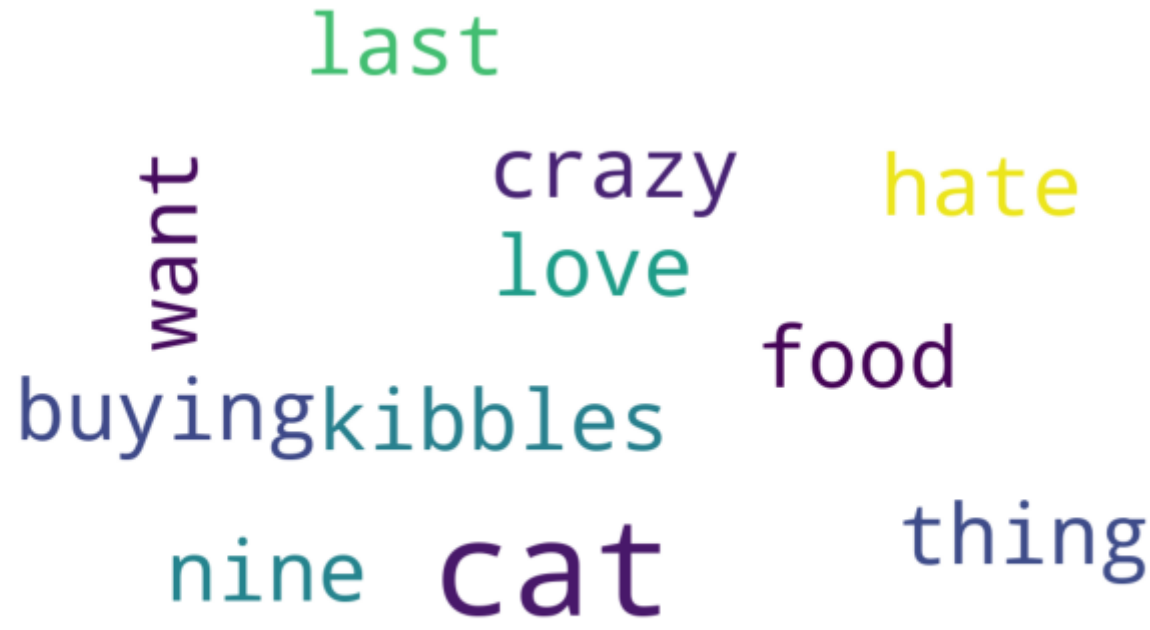


cluster 1

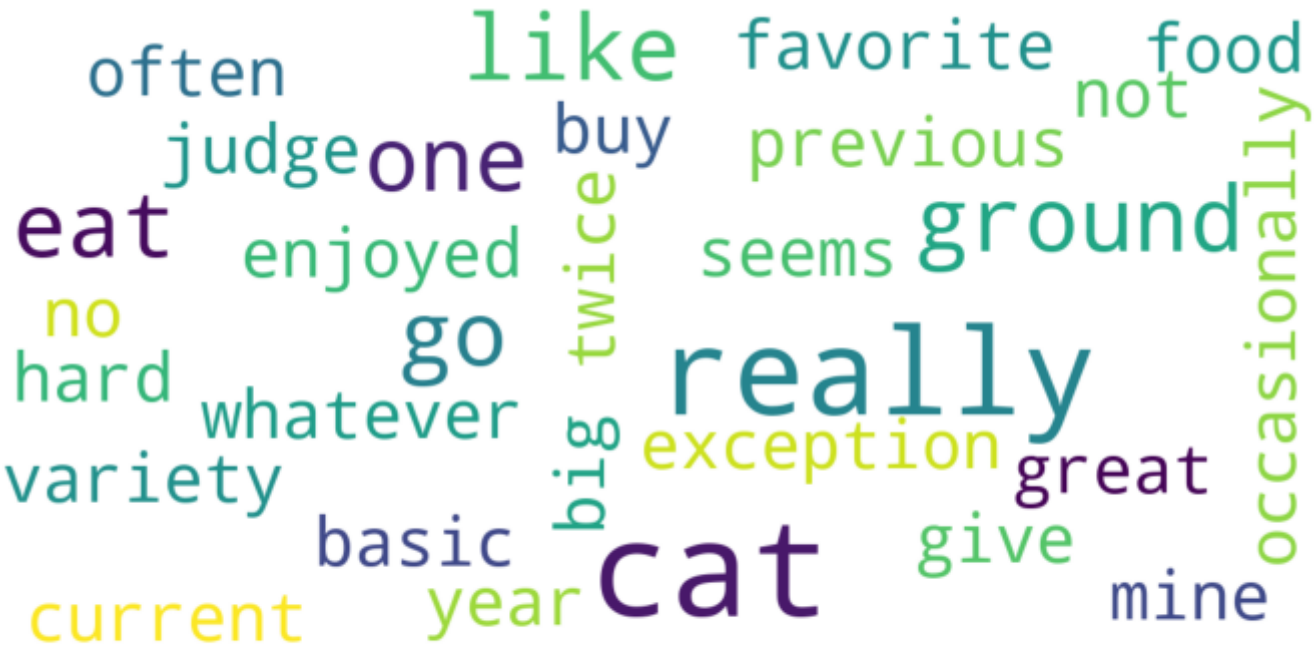
cluster 1

cluster 2

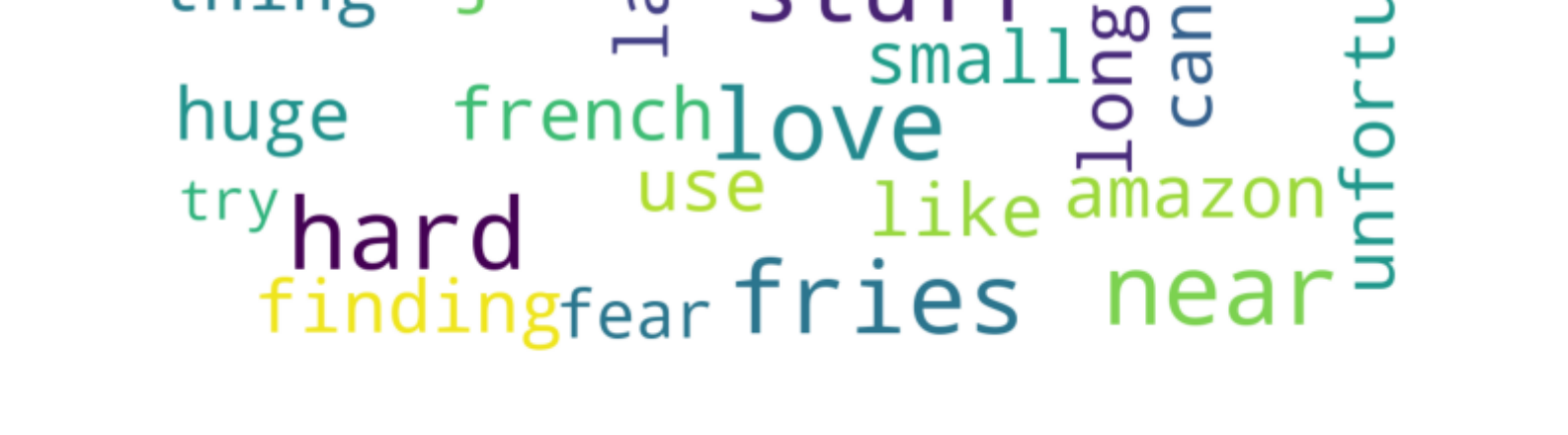
cluster 2



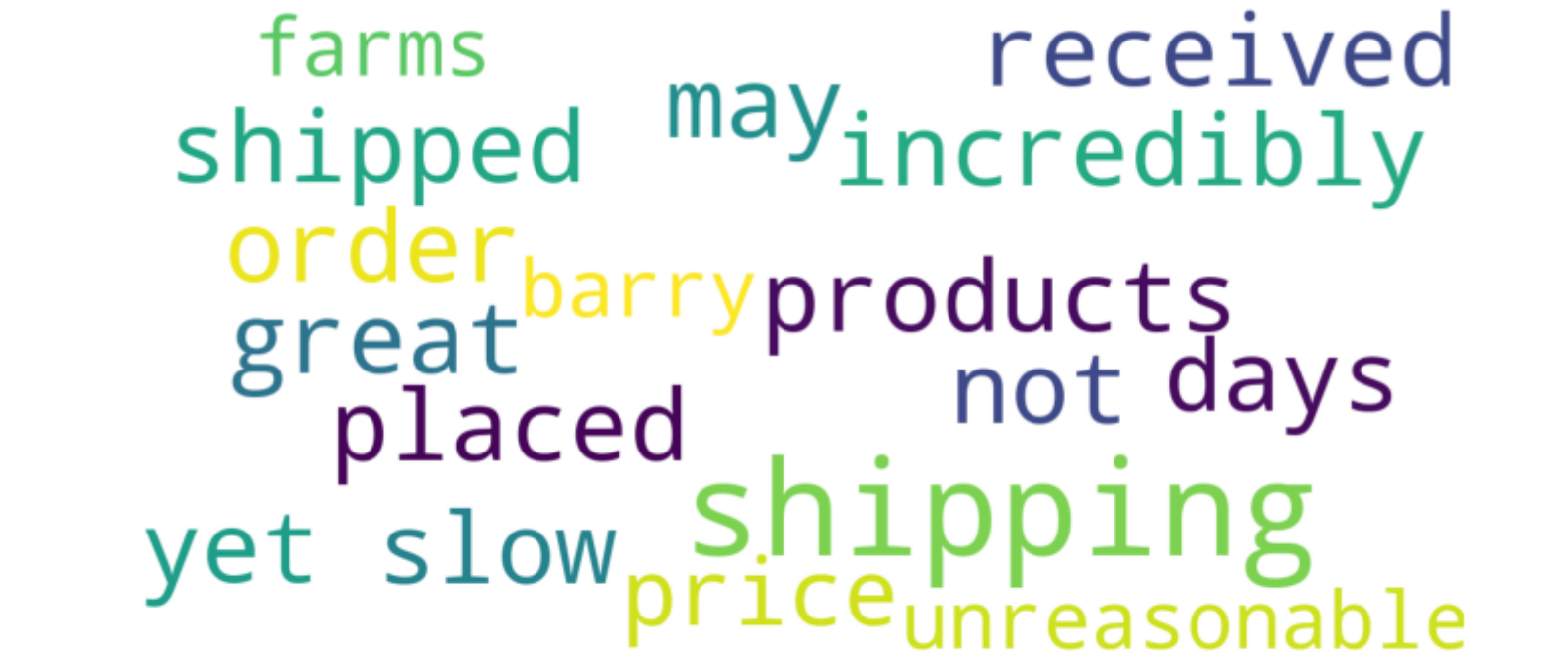
cluster 3



cluster 3



cluster 4



cluster 4

```
pt2=PrettyTable()


pt2.add_column("dbscan with",["avg_w2v","weighted_tfidf"])
pt2.add_column("min samples per cluester",[100,100])
pt2.add_column("epsilon",[[0.6,1.0,1.25,1.5],[0.5,1.3,1.4,1.5]])
pt2.add_column("cluster",[[(-1),(-1,0),(-1,0),(-1,0)],[(-1),(-1,0),(-1,0),(-1,0)]])
print(pt2)
```

```
+---------------+--------------------------+----------------------+-----------------
|  dbscan with  | min samples per cluester |       epsilon        |            clust
+---------------+--------------------------+----------------------+-----------------
|    avg_w2v    |           100            | [0.6, 1.0, 1.25, 1.5] | [-1, (-1, 0), (-1
| weighted_tfidf |           100            |  [0.5, 1.3, 1.4, 1.5] | [-1, (-1, 0), (-1
+---------------+--------------------------+----------------------+-----------------
```