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 considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

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

from tqdm import tqdm
import os
```

```
C:\Users\Cyborg\Anaconda3\lib\site-packages\gensim\utils.py:860: UserWarning: detected Windows; al
iasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect(r'C:\Users\Cyborg\Applied
AI\assignment\Assignments_AFR_2018\database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", co
n)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 7000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (7000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	130386240

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	134697600
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	121901760

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [3]:

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

In [4]:

```
sorted_data=sorted_data.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort'
, na_position='last')
```

In [5]:

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

Out[5]:

```
(6977, 10)
```

In [6]:

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

Out[6]:

```
99.67142857142856
```

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

In [7]:

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

In [8]:

```
#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()
```

```
(6977, 10)
```

Out[8]:

```
1    5826
0    1151
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

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

In [9]:

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

In [10]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [11]:

```
# 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', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
```

```
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [12]:

```
# 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%|██████████████████████████████████████████████████████████████████████████| 6977/6977
[00:05<00:00, 1341.85it/s]
```

[3.2] Preprocessing Review Summary

In [13]:

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

[4] Featurization

[4.1] BAG OF WORDS

[4.3] TF-IDF

[4.4] Word2Vec

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

[4.4.1.2] TFIDF weighted W2v

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

Clustering

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

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

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

   - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
   - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)
   - Apply agglomerative algorithm and try a different number of clusters like 2,5 etc.
   - Same as that of K-means, plot word clouds for each cluster and summarize in your own words what that cluster is representing.
   - You can take around 5000 reviews or so(as this is very computationally expensive one)

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

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

[5.1] K-Means Clustering

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

In [14]:

```
# Please write all the code with proper documentation
```

In [15]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
```

In [16]:

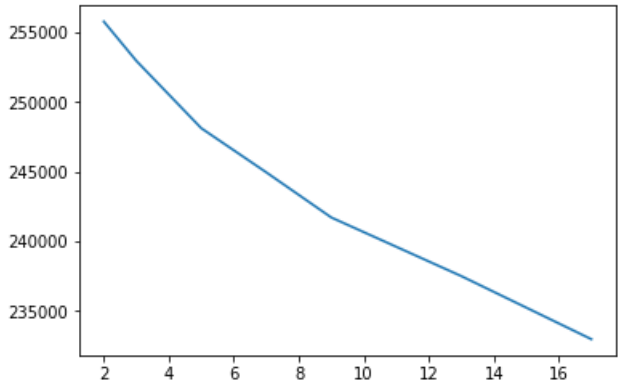
```
X=preprocessed_reviews[:]
y=final['Score'][:]
X_train, X_test, y_train, y_test = train_test_split(X,y , test_size=0.30, random_state=42)
bow = CountVectorizer()
X_train_bow=bow.fit_transform(X_train)
X_test_bow=bow.transform(X_test)
```

In [18]:

```
errors=[]
k_value=[2,3,5,7,9, 13, 17]
for i in k_value:
    kmeans = KMeans(n_clusters=i, random_state=0, n_jobs=-1)
    kmeans.fit(X_train_bow)
    pred=kmeans.predict(X_test_bow)
    errors.append(kmeans.inertia_)
plt.plot(k_value, errors)
```

Out[18]:

```
[<matplotlib.lines.Line2D at 0x1a4d28e1f98>]
```



In [19]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
model=KMeans(n_clusters=5, n_jobs=-1)
z=bow.fit_transform(preprocessed_reviews)
model.fit(scaler.fit_transform(z.toarray()))
```

```
C:\Users\Cyborg\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\Cyborg\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

Out[19]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=5, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [20]:

```
# df = preprocessed_reviews
# df['Bow Clus Label'] = model.labels_
# df.head(2)
w=model.labels_
```

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

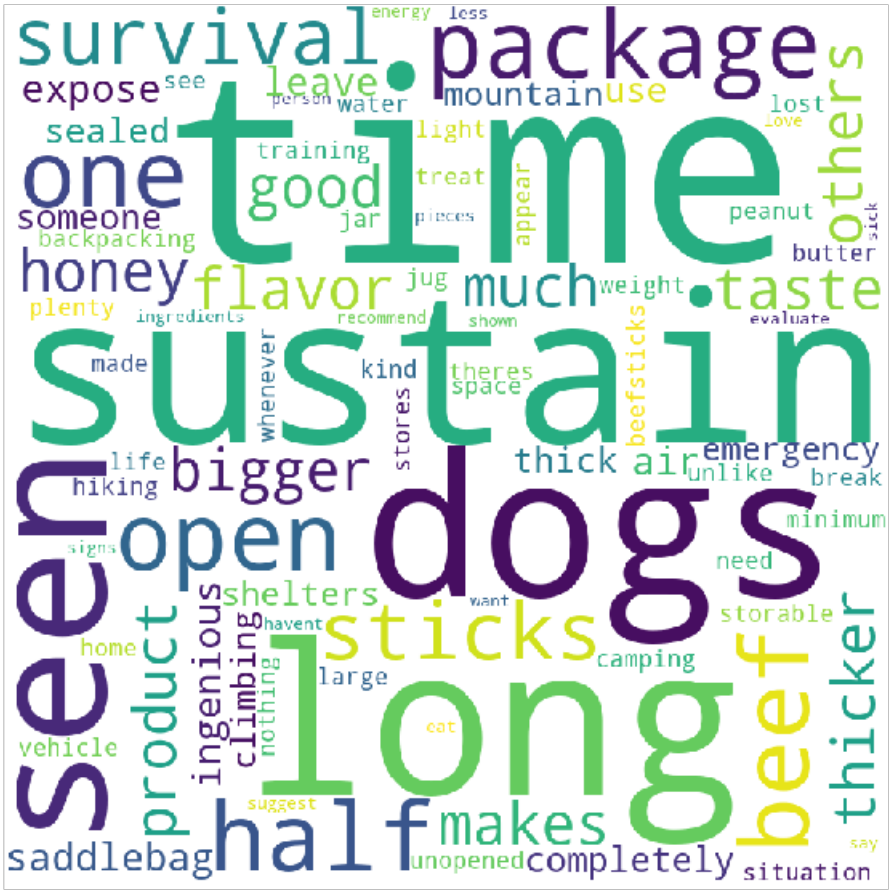
In [21]:

```
from wordcloud import WordCloud, STOPWORDS
```

In [22]:

```
bow_features=bow.get_feature_names()
# w is the weight after fitting the model
```

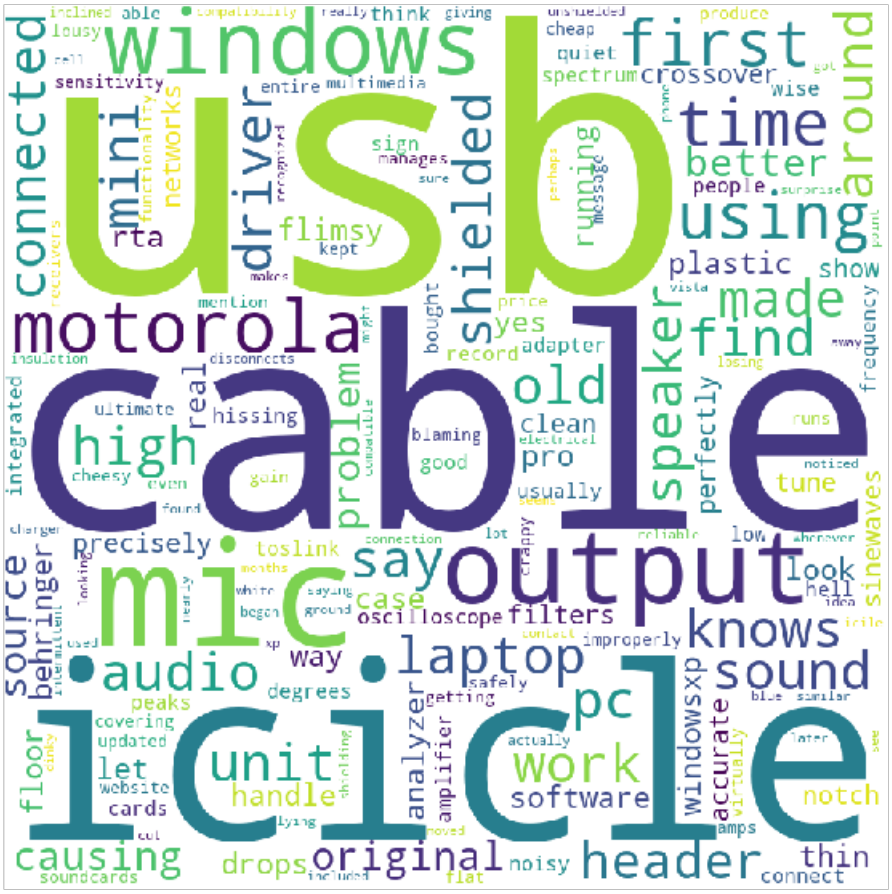
```
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



In [23]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[1:2]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()
```

```
    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



In [24]:

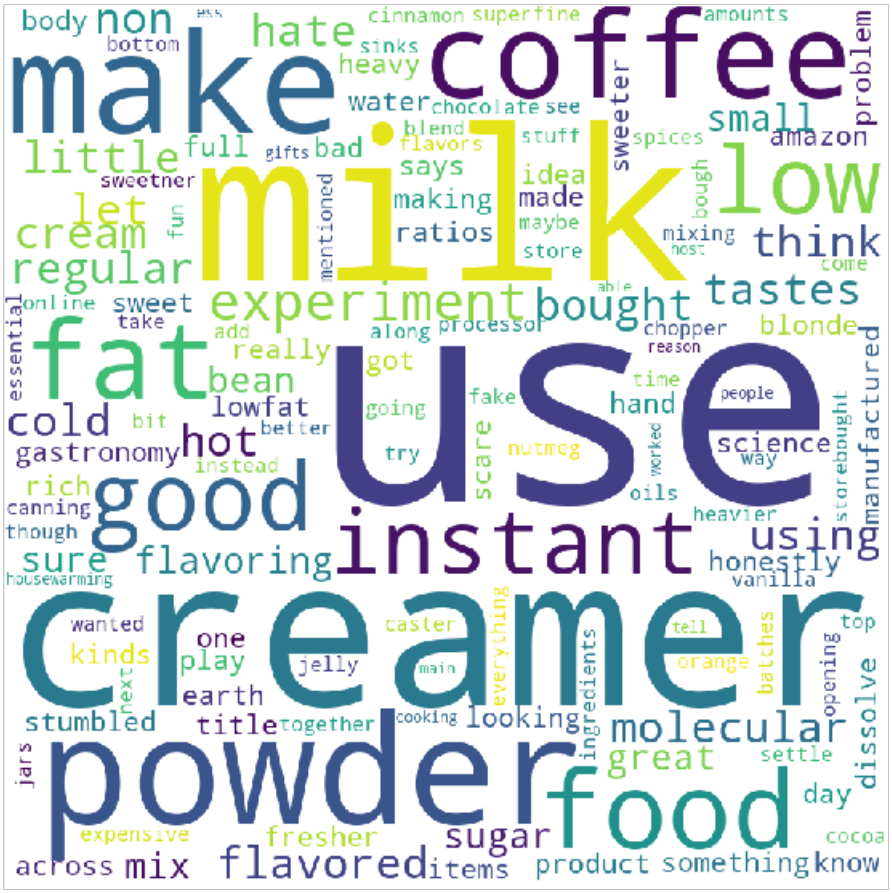
```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[2:3]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '
```

```
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [25]:

```
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[3:4]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
```

```
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



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

In [26]:

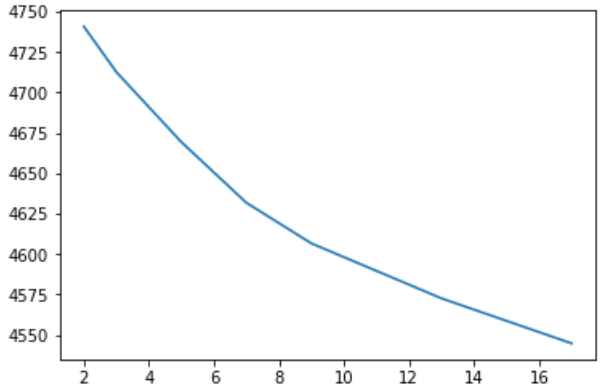
```
# Please write all the code with proper documentation
import warnings
warnings.filterwarnings("ignore")
```

In [27]:

```
X=preprocessed_reviews[:]
y=final['Score'][:]
X_train, X_test, y_train, y_test = train_test_split(X,y , test_size=0.30, random_state=42)
tfidf = TfidfVectorizer()
X_train_tfidf=tfidf.fit_transform(X_train)
X_test_tfidf=tfidf.transform(X_test)
tfidf_features=tfidf.get_feature_names()
```

In [29]:

```
errors=[]
k_value=[2,3,5,7,9, 13,  17]
for i in k_value:
    kmeans = KMeans(n_clusters=i, random_state=0, n_jobs=-1)
    kmeans.fit(X_train_tfidf)
    pred=kmeans.predict(X_test_tfidf)
    errors.append(kmeans.inertia_)
plt.plot(k_value, errors)
plt.show()
```



In [30]:

```
model=KMeans(n_clusters=5, n_jobs=-1)
z=bow.fit_transform(preprocessed_reviews)
model.fit(scaler.fit_transform(z.toarray()))
w=model.labels_
```

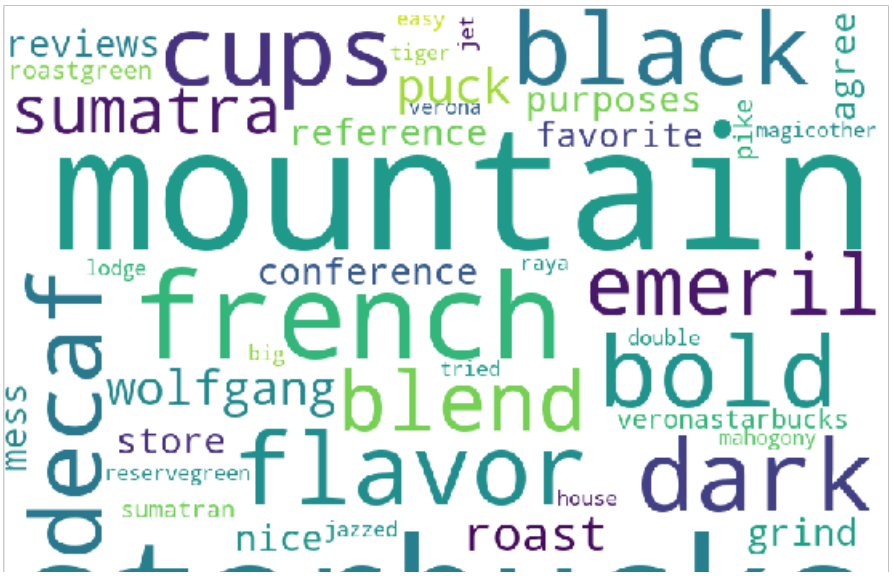
[5.1.4] Wordclouds of clusters obtained after applying k-means on TFIDF SET 2

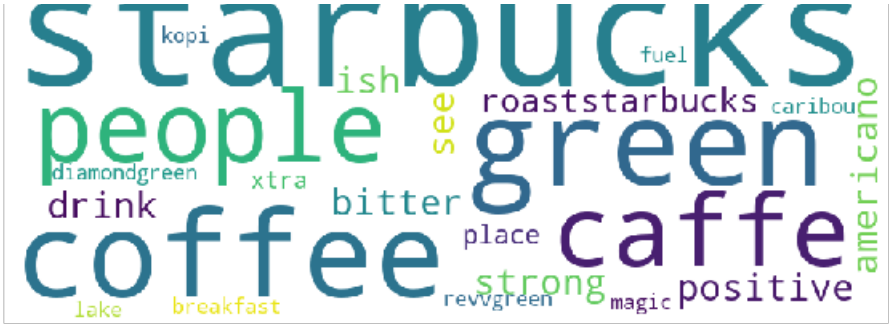
In [31]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [32]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[1:2]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```





In [33]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[2:3]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [34]:

```
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[3:4]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [35]:

In [35]:

```
# Please write all the code with proper documentation
```

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

In [36]:

```
# Please write all the code with proper documentation
```

In [37]:

```
from gensim.models import Word2Vec
import gensim
```

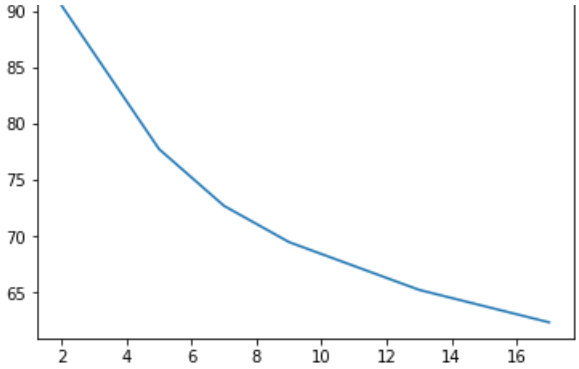
In [38]:

```
list_sent=[]
for sent in preprocessed_reviews:
    list_sent.append(sent.split())
X=preprocessed_reviews[:]
y=final['Score'][:]
X_train, X_test, y_train, y_test = train_test_split(X,y , test_size=0.30, random_state=42)

w2v_model=gensim.models.Word2Vec(X_train,min_count=5, size=50)
w2v_words = list(w2v_model.wv.vocab)
# Then vectorize your train model as
sent_vectors_train = [];
for sent in tqdm(X_train):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
# And then vectorize test as
sent_vectors_test = [];
for sent in tqdm(X_test):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 4883/4883
[00:04<00:00, 999.82it/s]
100%|██████████████████████████████████████████████████████████████████████████| 2094/2094
[00:02<00:00, 1044.92it/s]
```

In [39]:

```
errors=[]
k_value=[2,5,7,9,13, 17]
for i in k_value:
    kmeans = KMeans(n_clusters=i, random_state=0, n_jobs=-1)
    kmeans.fit(sent_vectors_train)
    pred=kmeans.predict(sent_vectors_test)
    errors.append(kmeans.inertia_)
plt.plot(k_value, errors)
plt.show()
```



In [40]:

```
model=KMeans(n_clusters=7, n_jobs=-1)
z=bow.fit_transform(preprocessed_reviews)
model.fit(scaler.fit_transform(z.toarray()))
w=model.labels_
```

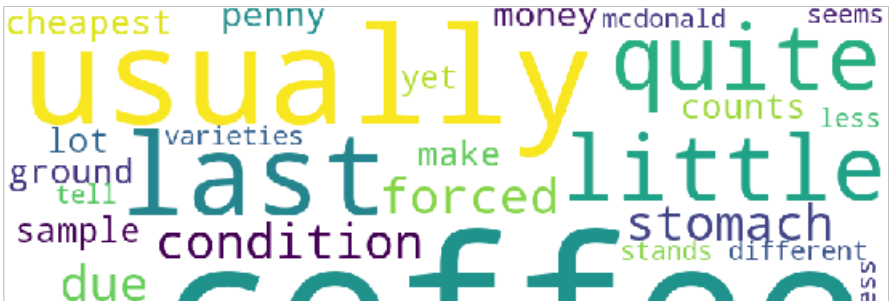
[5.1.6] Wordclouds of clusters obtained after applying k-means on AVG W2V SET 3

In [41]:

```
# Please write all the code with proper documentation
```

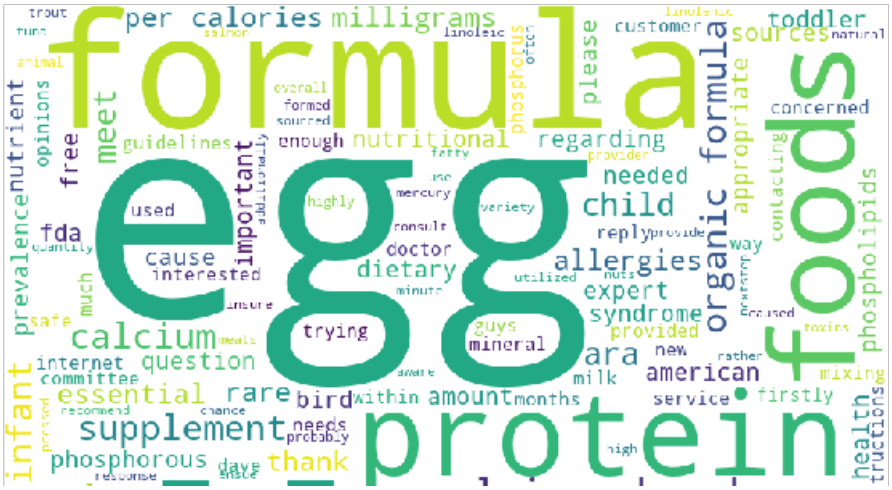
In [42]:

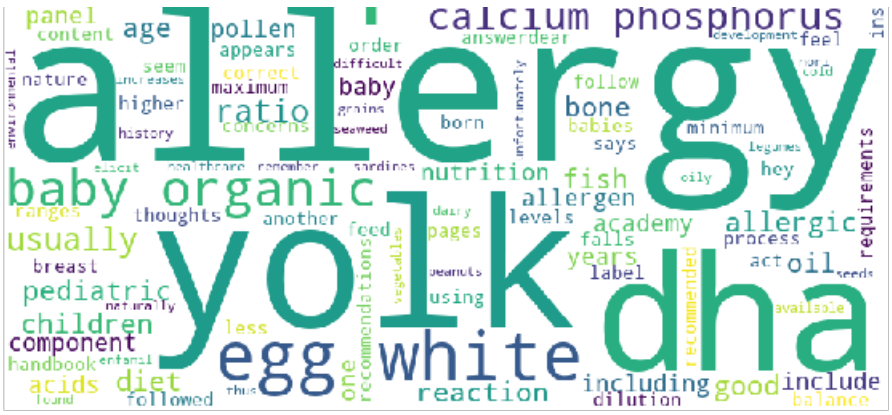
```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



In [43]:

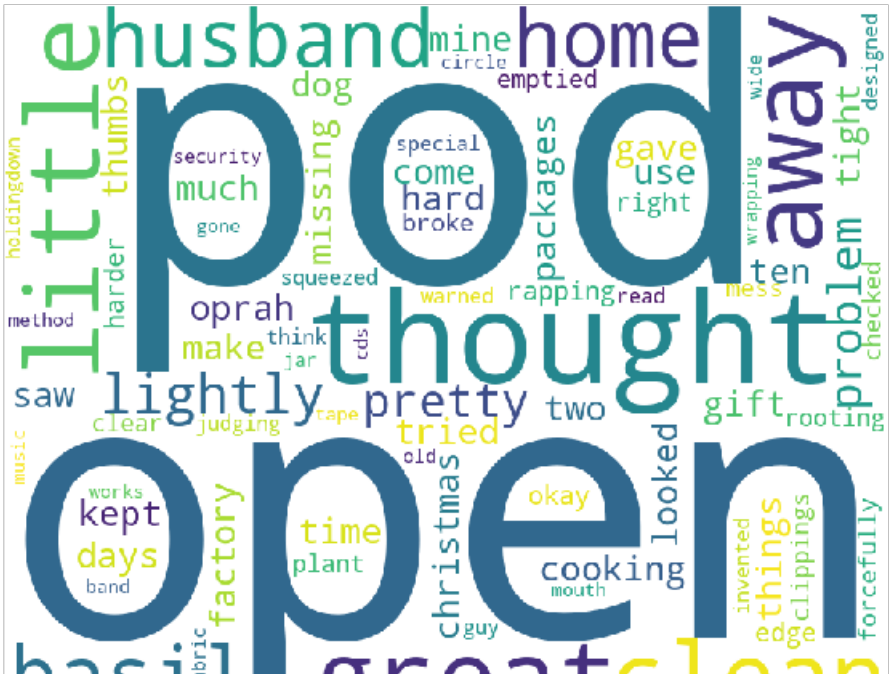
```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[1:2]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```





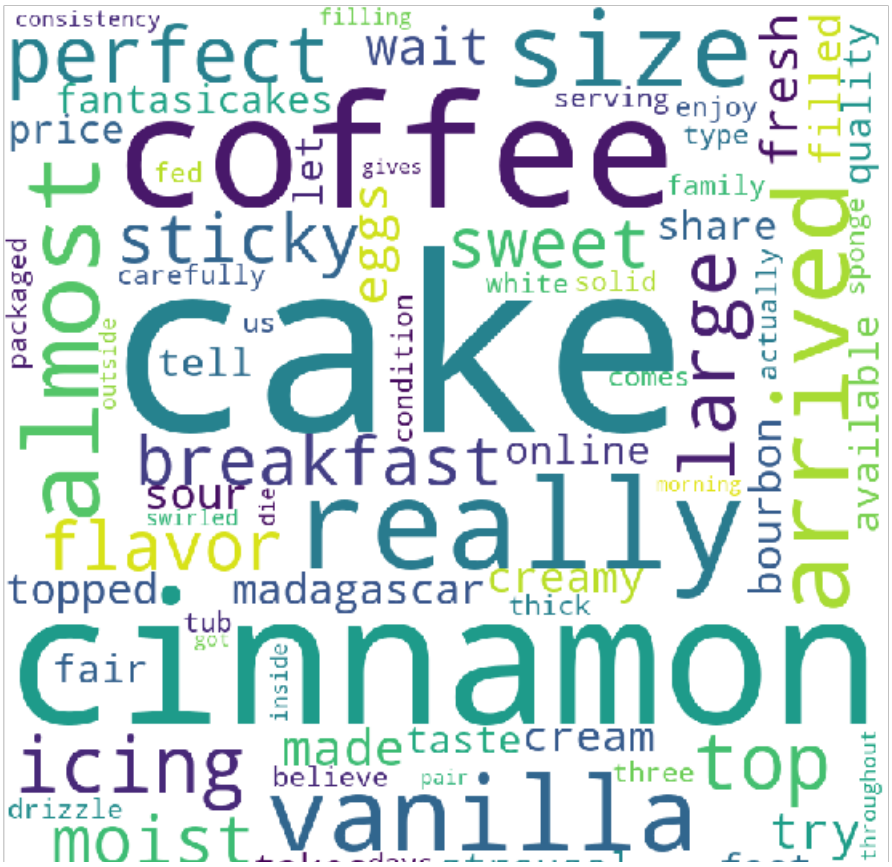
In [44]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[2:3]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



In [45]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[3:4]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



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

In [46]:

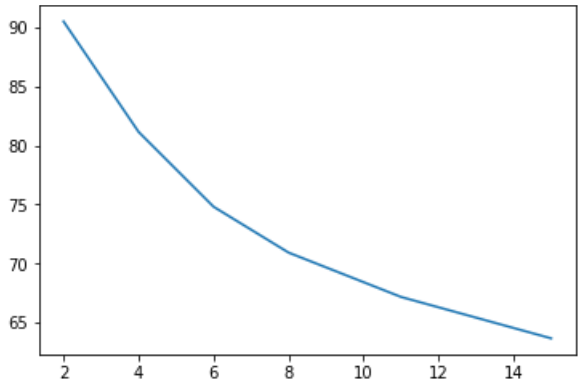
```
# Please write all the code with proper documentation
```

In [47]:

```
list_sent=[]
for sent in preprocessed_reviews:
    list_sent.append(sent.split())
X=list_sent[:]
y=final['Score'][:]
X_train, X_test, y_train, y_test = train_test_split(X,y , test_size=0.30, random_state=42)
w2v_model=gensim.models.Word2Vec(X_train,min_count=5, size=50)
w2v_words = list(w2v_model.wv.vocab)
dictionary = dict(zip(tfidf.get_feature_names(), list(tfidf.idf_)))
tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in X_train:
    # 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_features:
            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_train.append(sent_vec)
    row += 1

dictionary = dict(zip(tfidf.get_feature_names(), list(tfidf.idf_)))
tfidf_sent_vectors_test = [];

row=0;
for sent in X_test:
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_features:
            vec = w2v_model.wv[word]
            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_test.append(sent_vec)
    row += 1
```

In [48]:

```
errors=[]
k_value=[2,4,6,8,11, 15]
for i in k_value:
    kmeans = KMeans(n_clusters=i, random_state=0, n_jobs=-1)
    kmeans.fit(sent_vectors_train)
    pred=kmeans.predict(sent_vectors_test)
    errors.append(kmeans.inertia_)
plt.plot(k_value, errors)
plt.show()
```



In [50]:

```
model=KMeans(n_clusters=5, n_jobs=-1)
z=bow.fit_transform(preprocessed_reviews)
model.fit(scaler.fit_transform(z.toarray()))
w=model.labels_
```

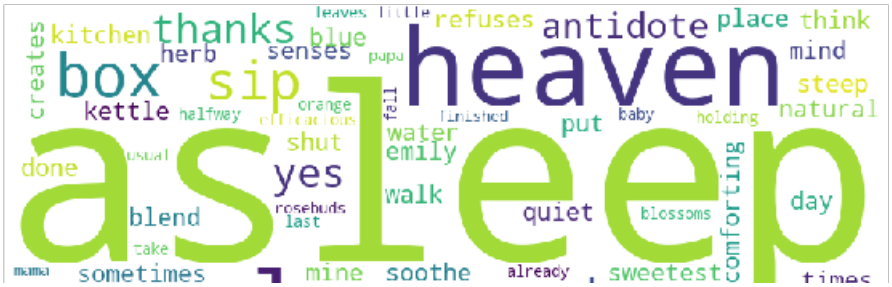
[5.1.8] Wordclouds of clusters obtained after applying k-means on TFIDF W2V SET 4

In [51]:

```
# Please write all the code with proper documentation
```

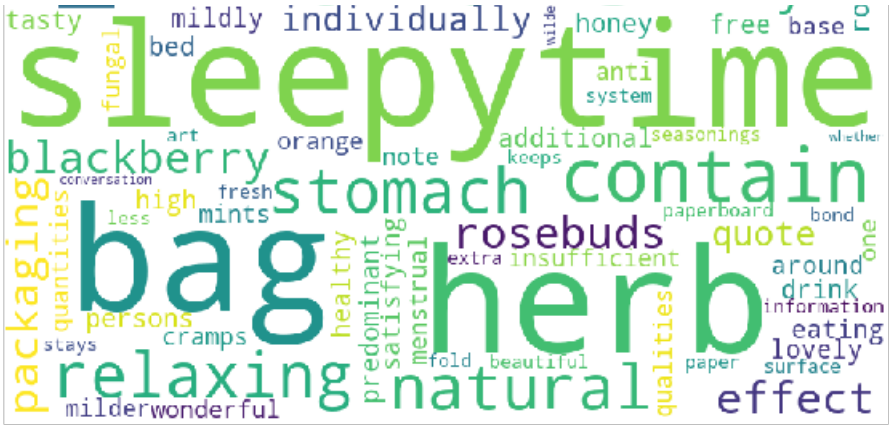
In [52]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



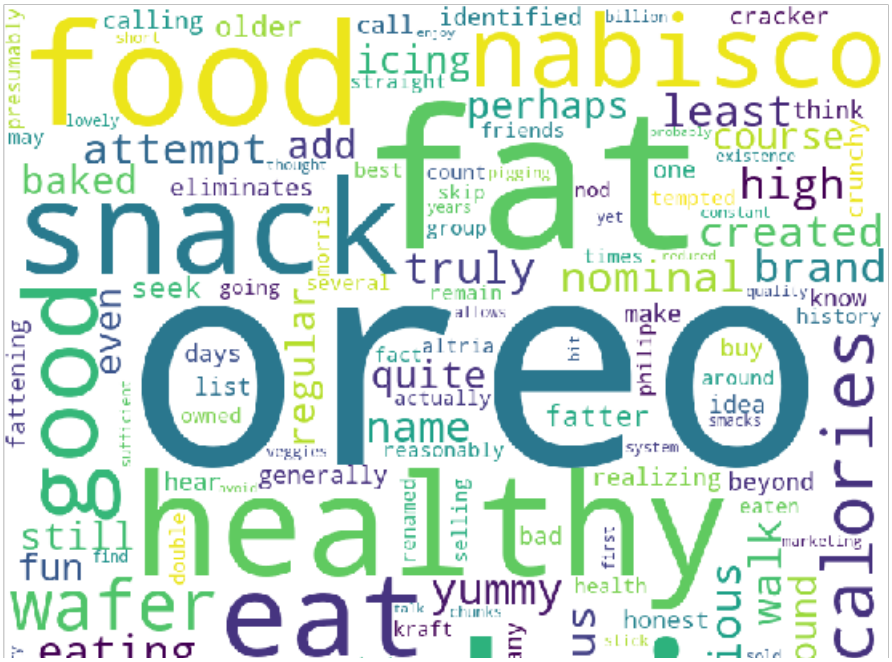
In [53]:

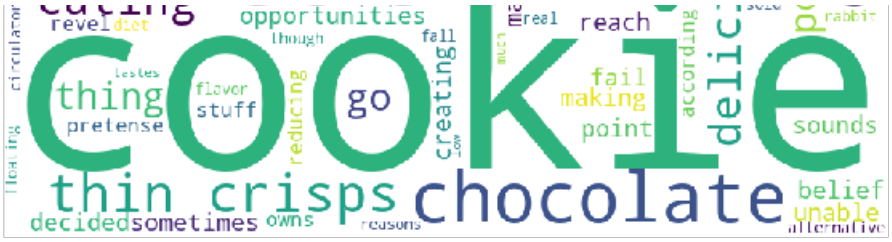
```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[1:2]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



In [54]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[2:3]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```





In [55]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews, columns=['importance']).sort_val
ues('importance',ascending=False)
a=feature_importances.iloc[3:4]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

Note:

1. Agglomerative clustering on 2k datapoints
2. DBSCAN on 2k data
3. KMeans on 7k data.
4. At some places the number of clusters may be large(upto 7 or 8), so printed a subset of that. But since it is represented by wordcloud, it makes it very interpretable

[5.2] Agglomerative Clustering

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

In [56]:

```
# Please write all the code with proper documentation
from sklearn.cluster import AgglomerativeClustering
```

In [57]:

```
# model=KMeans(n_clusters=7, n_jobs=-1)
model = AgglomerativeClustering(n_clusters=4)
sent_vectors_train=sent_vectors_train[:2000]
model.fit(scaler.fit_transform(sent_vectors_train))
w=model.labels_
```

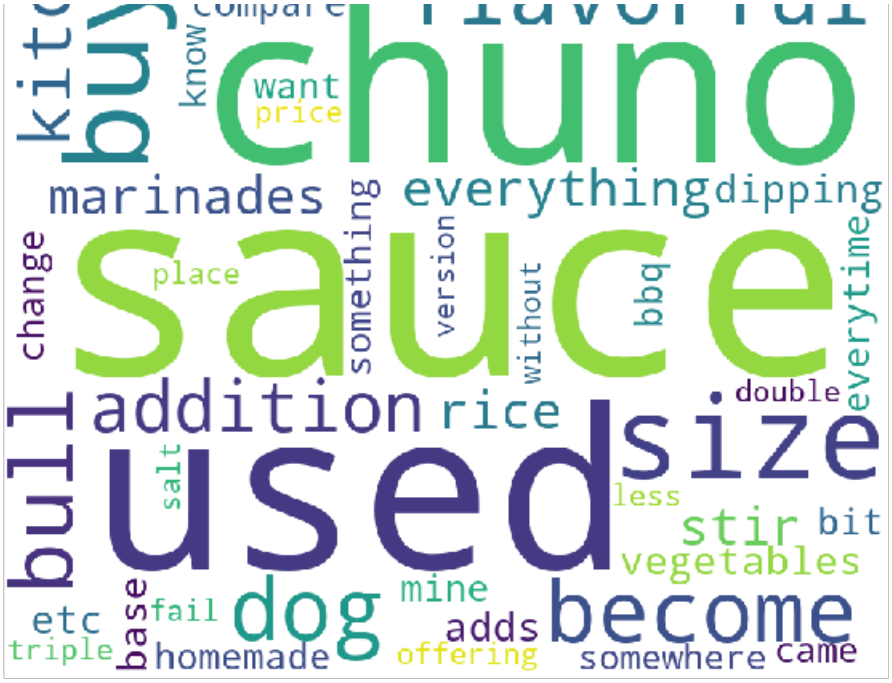
[5.2.2] Wordclouds of clusters obtained after applying Agglomerative Clustering on AVG W2V SET 3

In [58]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

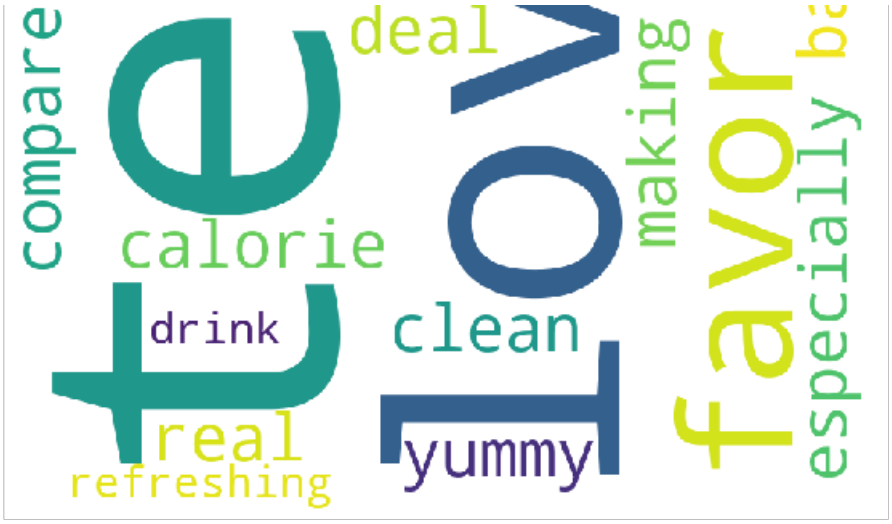
In [59]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[1:2]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



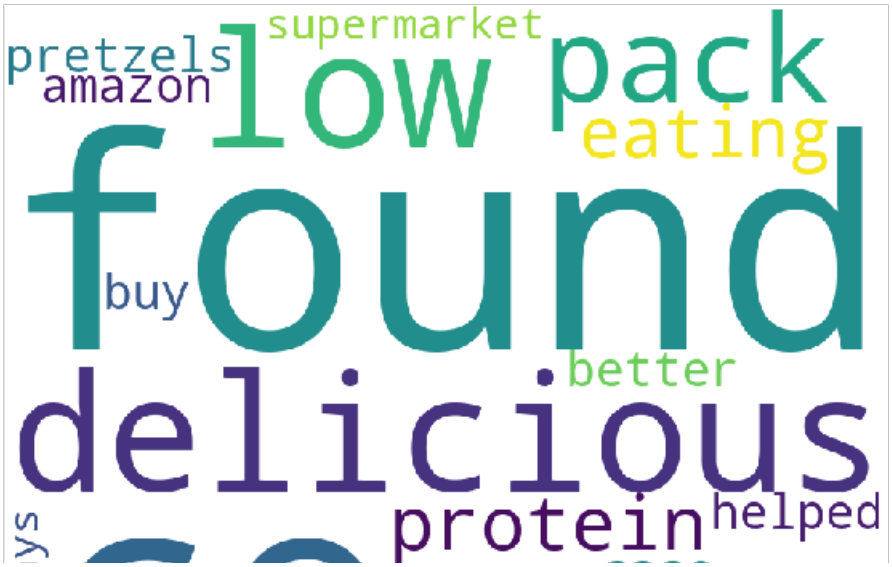
In [60]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[2:3
                           ]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
#
plt.show()
```



In [61]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[3:4]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



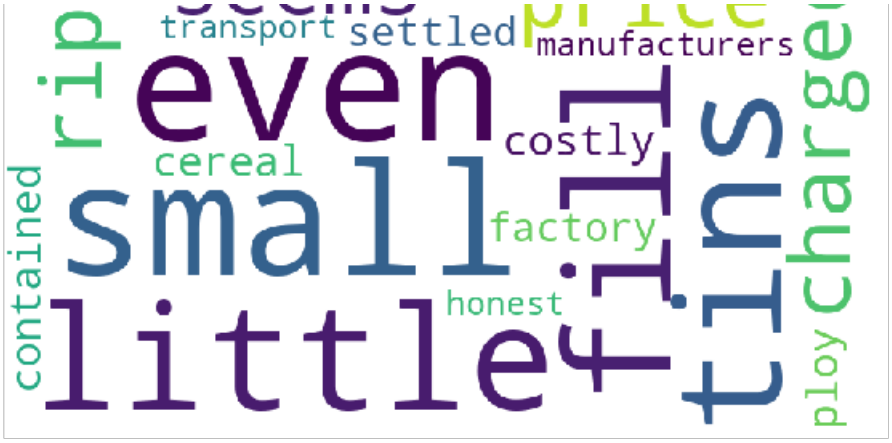
For different no of clusters and word cloud

In [62]:

```
model = AgglomerativeClustering(n_clusters=5)
model.fit(scaler.fit_transform(sent_vectors_train))
w=model.labels_
```

In [60]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

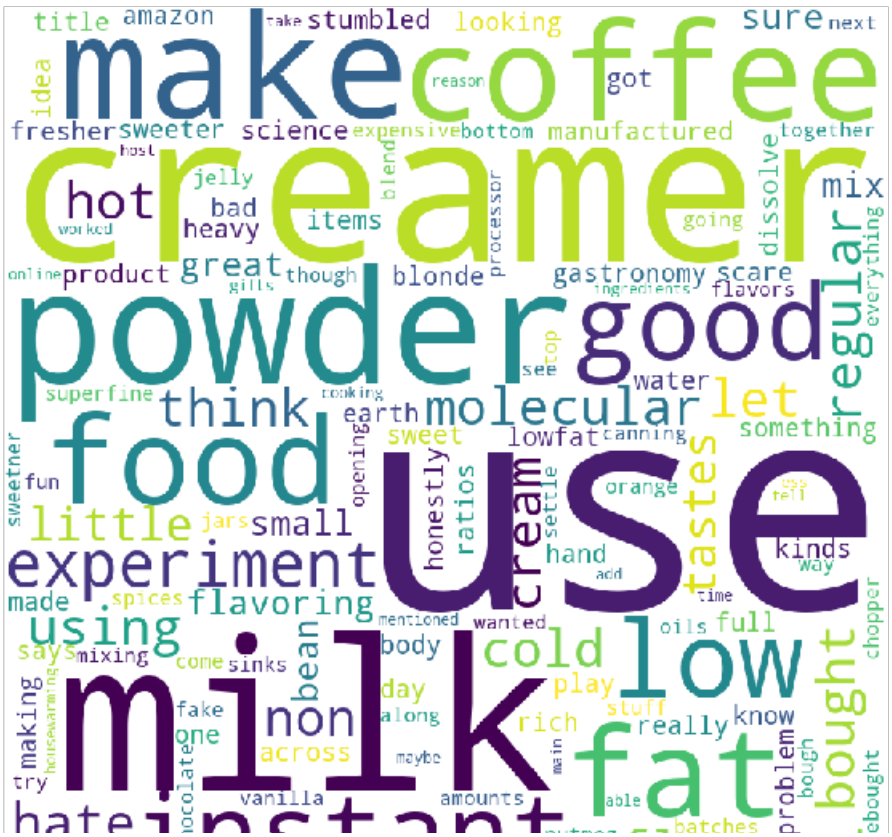


In [61]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[1:2]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [62]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[2:3]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



In [63]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[3:4]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [64]:

```
# Please write all the code with proper documentation
```

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

In [65]:

```
# Please write all the code with proper documentation
```

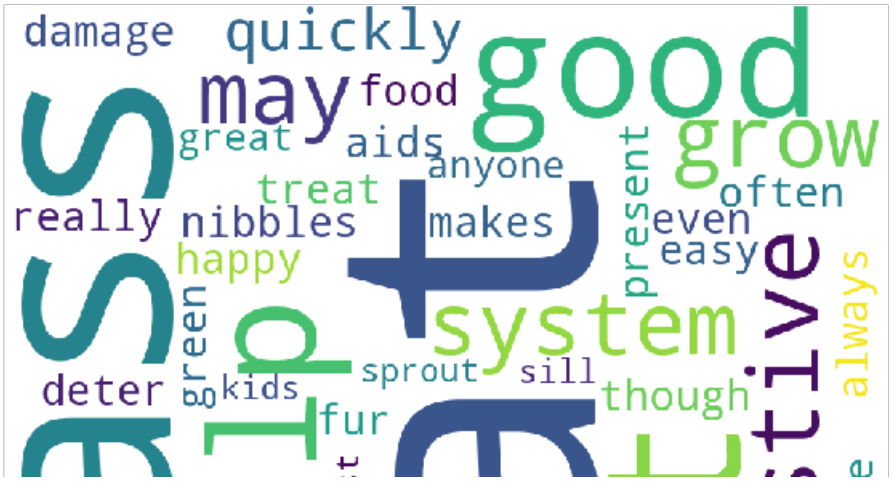
In [66]:

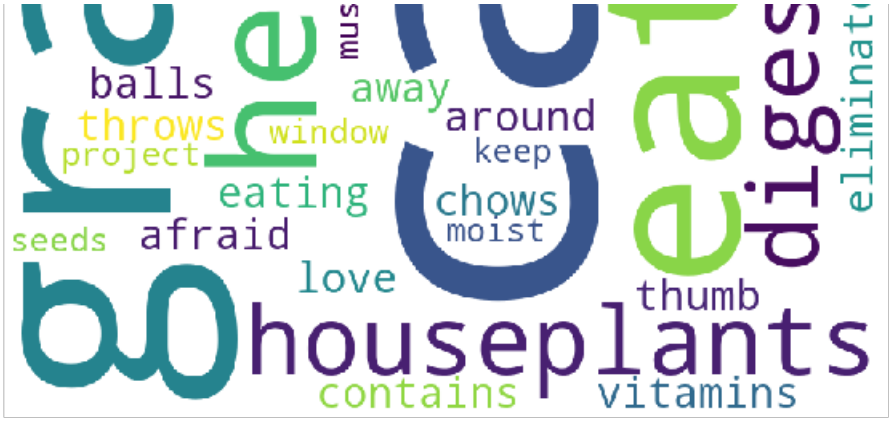
```
# model=KMeans(n_clusters=7, n_jobs=-1)
model = AgglomerativeClustering(n_clusters=4)
tfidf_sent_vectors_train=tfidf_sent_vectors_train[0:2000]
model.fit(scaler.fit_transform(tfidf_sent_vectors_train))
w=model.labels_
```

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

In [67]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```





In [68]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[1:2]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [69]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[2:3]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

For some different value of n_clusters

In [70]:

```
model = AgglomerativeClustering(n_clusters=5)
tfidf_sent_vectors_train=tfidf_sent_vectors_train[0:2000]
model.fit(scaler.fit_transform(tfidf_sent_vectors_train))
w=model.labels_
```

In [71]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [72]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[1:2]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [73]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[2:3]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [70]:

```
# Please write all the code with proper documentation
```

[5.3] DBSCAN Clustering

[5.3.1] Applying DBSCAN on AVG W2V, SET 3

In [71]:

```
# Please write all the code with proper documentation
```

In [77]:

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

In [78]:

```
model = DBSCAN(eps=1)
model.fit(scaler.fit_transform(sent_vectors_train))
w=model.labels_
```

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

In [79]:

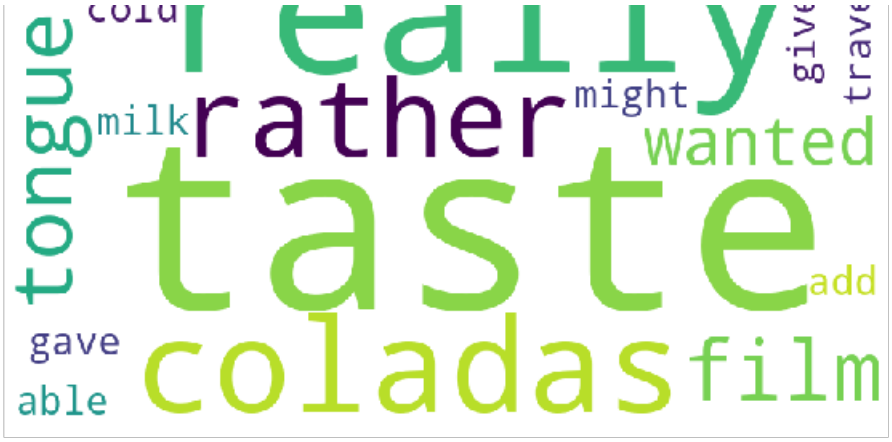
```
# Please write all the code with proper documentation
```

In [80]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [81]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[1:2]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



In [82]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[2:3]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

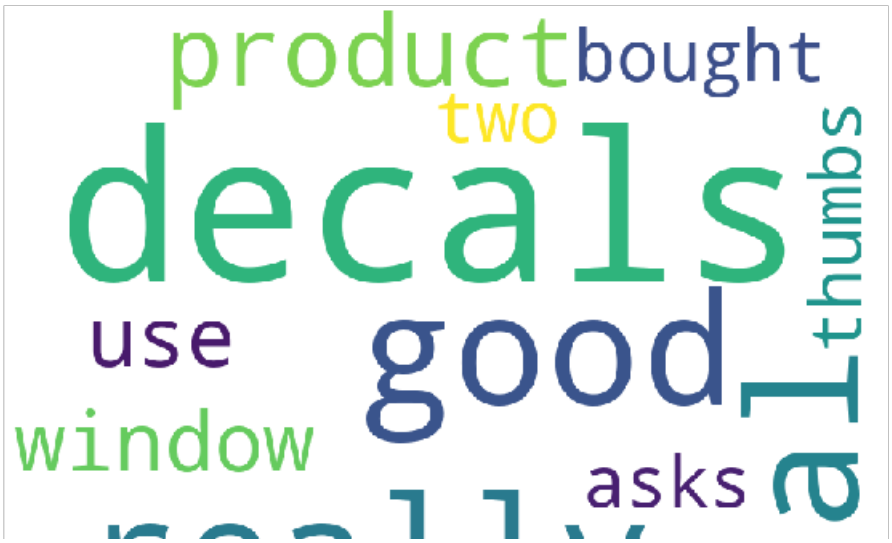
For different value of eps

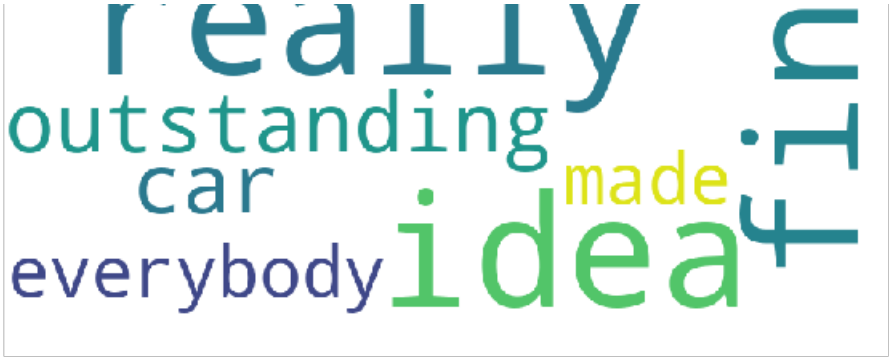
In [83]:

```
model = DBSCAN(eps=0.5)
model.fit(scaler.fit_transform(sent_vectors_train))
w=model.labels_
```

In [84]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```





In [85]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[2:3]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

[5.3.3] Applying DBSCAN on TFIDF W2V, SET 4

In [86]:

```
# Please write all the code with proper documentation
```

In [87]:

```
model = DBSCAN(eps=1)
model.fit(scaler.fit_transform(tfidf_sent_vectors_train))
w=model.labels_
```

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

In [88]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [89]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[1:2]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [90]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[2:3]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [91]:

```
# Please write all the code with proper documentation
```

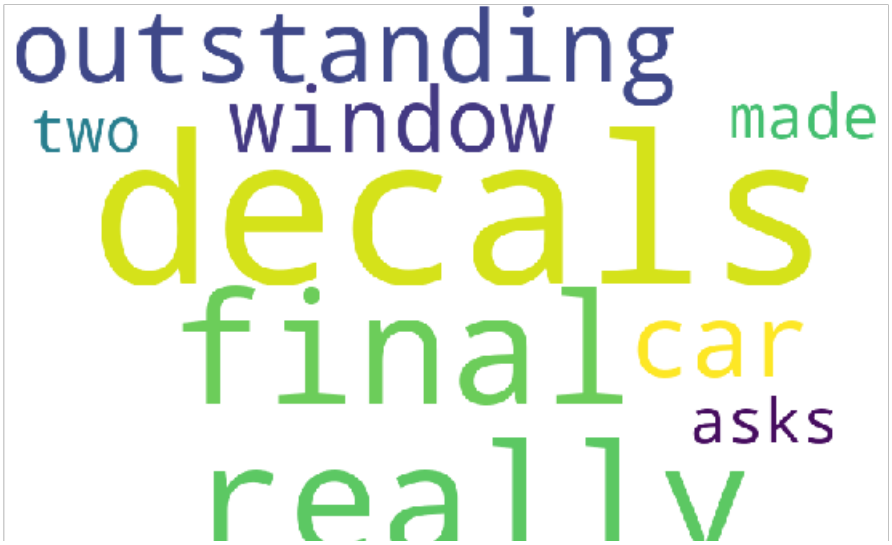
For some other value of eps

In [92]:

```
model = DBSCAN(eps=0.5)
model.fit(scaler.fit_transform(tfidf_sent_vectors_train))
w=model.labels_
```

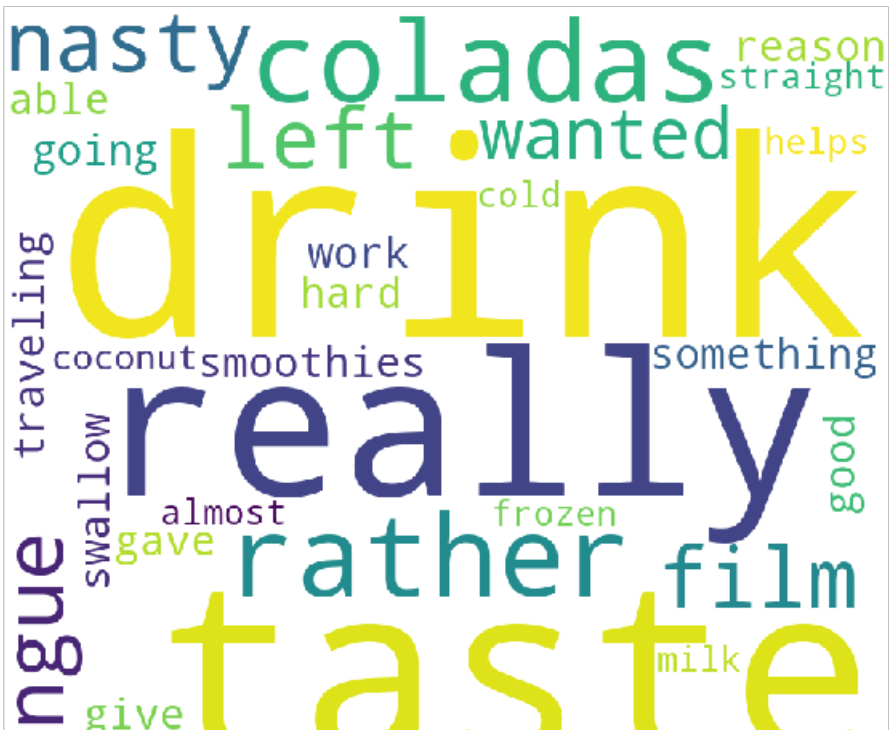
In [93]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[0:1]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



In [94]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[1:2]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



In [95]:

```
# bow_features=bow.get_feature_names()
# w is the weight after fitting the model
feature_importances = pd.DataFrame(w,index = preprocessed_reviews[:len(w)], columns=['importance'])
.sort_values('importance',ascending=False)
a=feature_importances.iloc[2:3]
comment_words = ' '
for val in a.index:
    val = str(val)
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

        stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

[6] Conclusions

In [96]:

```
# Please compare all your models using Prettytable library.
# You can have 3 tables, one each for kmeans, agllomerative and dbscan
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer","Cluster", "optimal clusters"]

x.add_row(["BOW" ,"KMeans" ,5])
x.add_row(["TFIDF" ,"KMeans", 5])
x.add_row(["Avgw2vec", "KMeans",  7])
x.add_row(["TFIDF-W2V",  "KMeans",  5])
print(x)
x = PrettyTable()

x.field_names = ["Vectorizer","Cluster", "no. of clusters"]

x.add_row(["Avgw2vec", "Agglomerative",  '4 and 5'])
x.add_row(["TFIDF-W2V",  "Aggolomerative",  '4 and 5'])
print(x)

x = PrettyTable()

x.field_names = ["Vectorizer","Cluster", "eps"]

x.add_row(["Avgw2vec", "DBSCAN",  '0.5 and 1'])
x.add_row(["TFIDF-W2V",  "DBSCAN",  '0.5 and 1'])
print(x)
```

```
+------------+---------+------------------+
| Vectorizer | Cluster | optimal clusters |
+------------+---------+------------------+
|    BOW     |  KMeans |        5         |
|   TFIDF    |  KMeans |        5         |
|  Avgw2vec  |  KMeans |        7         |
| TFIDF-W2V  |  KMeans |        5         |
+------------+---------+------------------+
+------------+----------------+-----------------+
| Vectorizer |    Cluster     | no. of clusters |
+------------+----------------+-----------------+
|  Avgw2vec  | Agglomerative  |     4 and 5     |
| TFIDF-W2V  | Aggolomerative |     4 and 5     |
+------------+----------------+-----------------+
+------------+---------+-----------+
| Vectorizer | Cluster |    eps    |
+------------+---------+-----------+
|  Avgw2vec  |  DBSCAN | 0.5 and 1 |
| TFIDF-W2V  |  DBSCAN | 0.5 and 1 |
+------------+---------+-----------+
```