SUYASH PRATAP SINGH(181B226)

TASKS:-

1. Download the dataset for Amazon Fine Food Reviews from Kaggle.
2. Subset it to contain only text field.
3. Apply LSA on that field to categorize it into 10 topics.
4. Visualize those topics using woordcloud.

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
#configure
# sets matplotlib to inline and displays graphs below the corressponding cell.
%matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid',color_codes=True)

#import nltk
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize,sent_tokenize

#preprocessing
from nltk.corpus import stopwords  #stopwords
from nltk import word_tokenize,sent_tokenize # tokenizing
from nltk.stem import PorterStemmer,LancasterStemmer  # using the Porter Stemmer and Lancaster Stemmer and others
from nltk.stem.snowball import SnowballStemmer
from nltk.stem import WordNetLemmatizer  # lammatizer from WordNet

# for named entity recognition (NER)
from nltk import ne_chunk

# vectorizers for creating the document-term-matrix (DTM)
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer

#stop-words
stop_words=set(nltk.corpus.stopwords.words('english'))
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/amazon-fine-food-reviews/hashes.txt
/kaggle/input/amazon-fine-food-reviews/Reviews.csv
/kaggle/input/amazon-fine-food-reviews/database.sqlite
```

In [2]:
```
df = pd.read_csv('/kaggle/input/amazon-fine-food-reviews/Reviews.csv')
df.head()
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomi
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	

In [3]:
```
print(df.shape)
print(df.isnull().values.any())
```

```
(568454, 10)
True
```

In [4]:
```
df.drop(['Id','ProductId','UserId','ProfileName','HelpfulnessNumerator','Helpf
ulnessDenominator','Score','Time','Summary'],axis=1,inplace=True)
```

In [5]:
```
df.head()
```

Out[5]:

	Text
0	I have bought several of the Vitality canned d...
1	Product arrived labeled as Jumbo Salted Peanut...
2	This is a confection that has been around a fe...
3	If you are looking for the secret ingredient i...
4	Great taffy at a great price. There was a wid...

In [6]:
```
def clean_text(headline):
  le=WordNetLemmatizer()
  word_tokens=word_tokenize(headline)
  tokens=[le.lemmatize(w) for w in word_tokens if w not in stop_words and len(
w)>3]
  cleaned_text=" ".join(tokens)
  return cleaned_text
```

In [7]:
```
df['Text']=df['Text'].apply(clean_text)
```

In [8]:
```
vect =TfidfVectorizer(stop_words=stop_words,max_features=1000,ngram_range = (1
,3)) # to play with. min_df,max_df,max_features etc...
vect_text=vect.fit_transform(df['Text'])
#vectorizer = TfidfVectorizer(ngram_range = (1,3))
```

In [9]:
```
print(vect_text.shape)
print(vect_text)
```

```
(568454, 1000)
  (0, 504)      0.30318304462156975
  (0, 115)      0.3162363190689679
  (0, 72)       0.3351026921211797
  (0, 799)      0.21821772496059946
  (0, 535)      0.2568130207133922
  (0, 681)      0.3184701536414067
  (0, 483)      0.11156894352481986
  (0, 503)      0.21921563658053317
  (0, 691)      0.20403716683578527
  (0, 365)      0.11967955811066665
  (0, 336)      0.18080901287346596
  (0, 682)      0.38084596642628005
  (0, 331)      0.16046827628977436
  (0, 114)      0.27422386993031245
  (0, 768)      0.22828365250331556
  (0, 90)       0.1831131220264846
  (1, 862)      0.3226831290792325
  (1, 792)      0.48082653289402927
  (1, 797)      0.32175021423903283
  (1, 10)       0.3368554946210655
  (1, 633)      0.39207415353666536
  (1, 45)       0.36999810052437265
  (1, 682)      0.39631538273241423
  (2, 409)      0.22961784148439382
  (2, 918)      0.36282717813408827
  :     :
  (568451, 25)  0.12391228067885442
  (568451, 325) 0.1013909254411724
  (568451, 518) 0.14536263185925224
  (568451, 442) 0.3179036192854717
  (568451, 918) 0.1498203400915397
  (568451, 797) 0.1608312782083553
  (568451, 799) 0.1702646620617786
  (568451, 483) 0.08705181244682576
  (568451, 331) 0.37561674019442354
  (568452, 917) 0.34426517695001463
  (568452, 515) 0.3286150587086632
  (568452, 866) 0.3745064798095839
  (568452, 663) 0.2971603155550557
  (568452, 111) 0.2502410240630782
  (568452, 756) 0.2800035351301658
  (568452, 71)  0.18518692075282894
  (568452, 865) 0.21362323417751952
  (568452, 513) 0.26150708026812364
  (568452, 304) 0.2144811527077717
  (568452, 918) 0.4370820359152258
  (568452, 365) 0.13621240378325528
  (568453, 952) 0.596365515344849
  (568453, 125) 0.4910334021522906
  (568453, 744) 0.5852114141737121
  (568453, 682) 0.24649943366876834
```

In [10]:
```
from sklearn.decomposition import TruncatedSVD
lsa_model = TruncatedSVD(n_components=10, algorithm='randomized', n_iter=10, r
andom_state=42)

lsa_top=lsa_model.fit_transform(vect_text)
```

In [11]:
```
print(lsa_top)
print(lsa_top.shape)  # (no_of_doc*no_of_topics)
```

```
[[ 2.44582460e-01 -1.23411914e-01  1.28098365e-01 ...  1.38027926e-01
   8.42362238e-02  1.53210359e-02]
 [ 1.36218396e-01 -7.94573968e-02  1.75130919e-02 ...  1.23242744e-01
   1.22555607e-01 -7.97698397e-02]
 [ 1.10956440e-01 -6.03839765e-02  3.82574440e-02 ... -1.63681793e-02
   3.66673064e-02 -5.58644608e-05]
 ...
 [ 2.59782369e-01 -1.62093827e-01  2.02888010e-01 ...  2.70714350e-03
  -2.96087192e-02 -1.46170860e-02]
 [ 1.51842314e-01 -7.66606248e-02  3.06543820e-02 ... -8.47479104e-02
   2.77411378e-03  1.34980178e-01]
 [ 7.19389561e-02 -4.64676887e-02 -6.09042003e-03 ...  7.85356648e-02
   4.91431831e-02 -2.64260840e-02]]
(568454, 10)
```

In [12]:
```
l=lsa_top[0]
print("Document 0 :")
for i,topic in enumerate(l):
  print("Topic ",i," : ",topic*100)
```

```
Document 0 :
Topic  0  :  24.458245976242853
Topic  1  :  -12.341191431524312
Topic  2  :  12.809836461838179
Topic  3  :  3.065953278009839
Topic  4  :  13.055016376057518
Topic  5  :  -10.567633778993024
Topic  6  :  -9.466065453526621
Topic  7  :  13.802792624080409
Topic  8  :  8.423622377662898
Topic  9  :  1.5321035944064456
```

In [13]:

```
vocab = vect.get_feature_names()

for i, comp in enumerate(lsa_model.components_):
    vocab_comp = zip(vocab, comp)
    sorted_words = sorted(vocab_comp, key= lambda x:x[1], reverse=True)[:10]
    print("Topic "+str(i)+": ")
    for t in sorted_words:
        print(t[0],end=" ")
    print("\n")
```

```
Topic 0:
coffee like taste good flavor great product love food would

Topic 1:
coffee strong cup roast bold blend keurig bitter cups drink

Topic 2:
food coffee treat dog cat love amazon product year price

Topic 3:
product amazon com amazon com www www amazon www amazon com http http www htt
p www amazon

Topic 4:
food like taste www http http www www amazon com www amazon http www amazon a
mazon com gp

Topic 5:
treat love dog like www www amazon www amazon com http http www http www amaz
on

Topic 6:
great food love chip flavor snack chocolate free gluten best

Topic 7:
great product taste great product taste great water food drink great taste re
commend

Topic 8:
chocolate product free cooky gluten gluten free good coffee bar peanut

Topic 9:
good chip price taste taste good really treat snack really good good price
```

In [14]:

```
Topics = []
for i, comp in enumerate(lsa_model.components_):
    vocab_comp = zip(vocab, comp)
    st=''
    sorted_words = sorted(vocab_comp, key= lambda x:x[1], reverse=True)[:10]
    for t in sorted_words:
        st = st + t[0] + " "
    Topics.append(st)
Topics
```

Out[14]:

```
['coffee like taste good flavor great product love food would ',
 'coffee strong cup roast bold blend keurig bitter cups drink ',
 'food coffee treat dog cat love amazon product year price ',
 'product amazon com amazon com www www amazon www amazon com http http www h
ttp www amazon ',
 'food like taste www http http www www amazon com www amazon http www amazon
amazon com gp ',
 'treat love dog like www www amazon www amazon com http http www http www am
azon ',
 'great food love chip flavor snack chocolate free gluten best ',
 'great product taste great product taste great water food drink great taste
recommend ',
 'chocolate product free cooky gluten gluten free good coffee bar peanut ',
 'good chip price taste taste good really treat snack really good good price
']
```

In [16]:

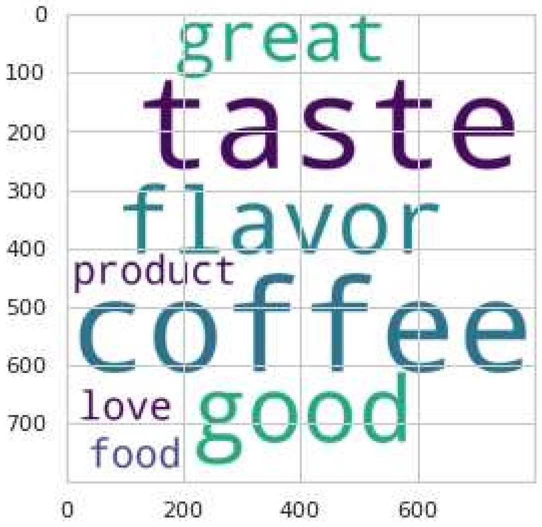
```
from wordcloud import WordCloud
```

In [18]:

```
vineet = WordCloud(background_color="white", width=800, height=800, random_sta
te=1).generate(Topics[0])
plt.imshow(vineet)
```

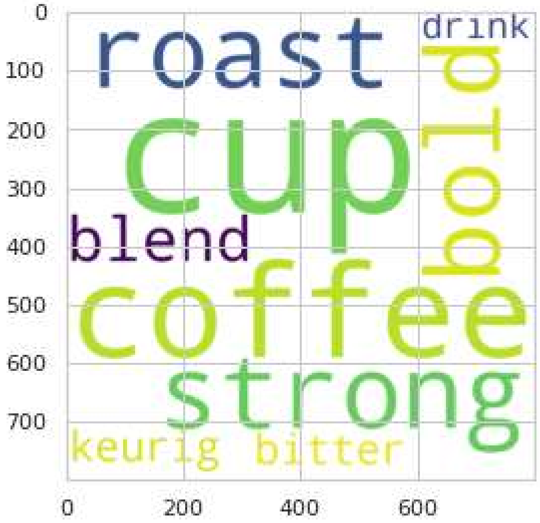
Out[18]:

```
<matplotlib.image.AxesImage at 0x7f8f49ea94d0>
```



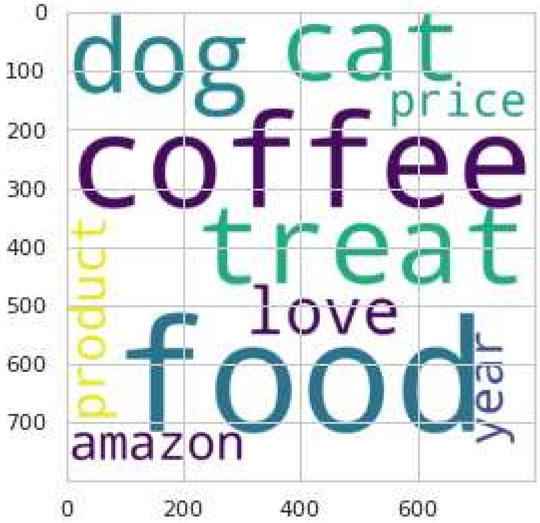
In [19]:
```
vineet = WordCloud(background_color="white", width=800, height=800, random_sta
te=1).generate(Topics[1])
plt.imshow(vineet)
```

Out[19]:
```
<matplotlib.image.AxesImage at 0x7f8f2a816e50>
```



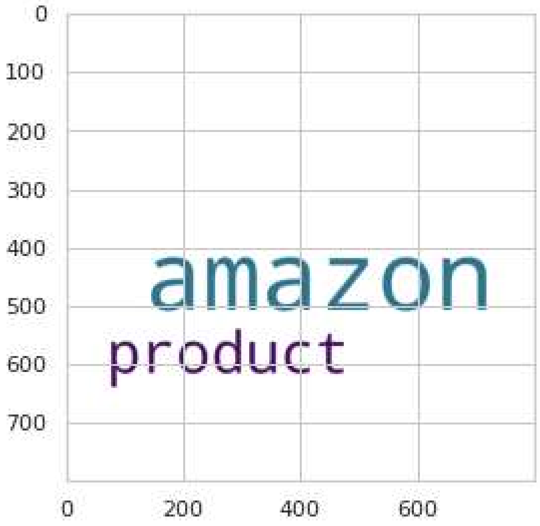
In [20]:
```
vineet = WordCloud(background_color="white", width=800, height=800, random_sta
te=1).generate(Topics[2])
plt.imshow(vineet)
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x7f8f44f16f90>
```



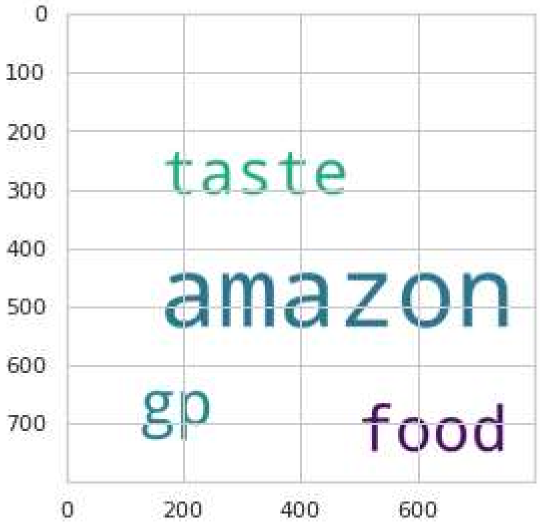
In [21]:
```
vineet = WordCloud(background_color="white", width=800, height=800, random_sta
te=1).generate(Topics[3])
plt.imshow(vineet)
```

Out[21]:
```
<matplotlib.image.AxesImage at 0x7f8f18519450>
```



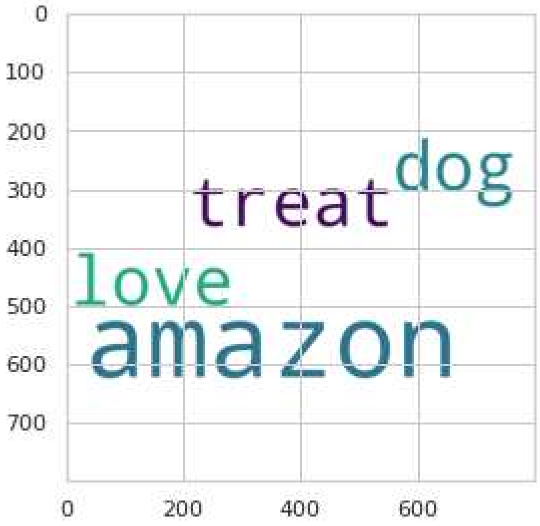
In [22]:
```
vineet = WordCloud(background_color="white", width=800, height=800, random_sta
te=1).generate(Topics[4])
plt.imshow(vineet)
```

Out[22]:
```
<matplotlib.image.AxesImage at 0x7f8f41a7f890>
```



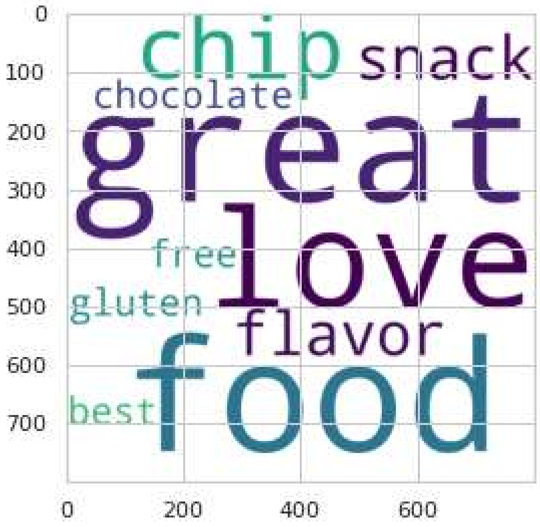
In [23]:
```
vineet = WordCloud(background_color="white", width=800, height=800, random_sta
te=1).generate(Topics[5])
plt.imshow(vineet)
```

Out[23]:
```
<matplotlib.image.AxesImage at 0x7f8f583f5c10>
```



In [24]:
```
vineet = WordCloud(background_color="white", width=800, height=800, random_sta
te=1).generate(Topics[6])
plt.imshow(vineet)
```

Out[24]:
```
<matplotlib.image.AxesImage at 0x7f8f20cc7b50>
```

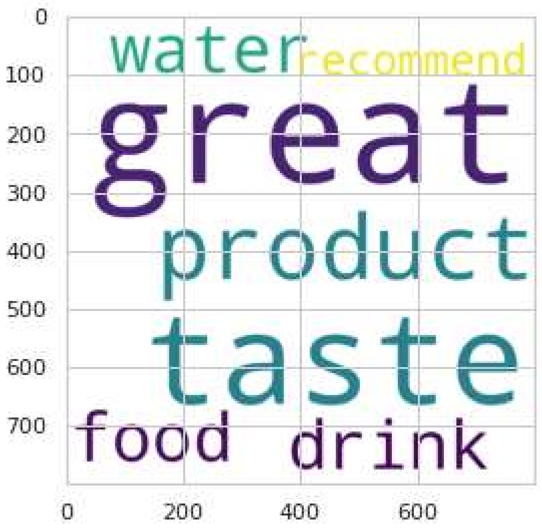


In [25]:

```
vineet = WordCloud(background_color="white", width=800, height=800, random_state=1).generate(Topics[7])
plt.imshow(vineet)
```

Out[25]:

```
<matplotlib.image.AxesImage at 0x7f8f42b76410>
```

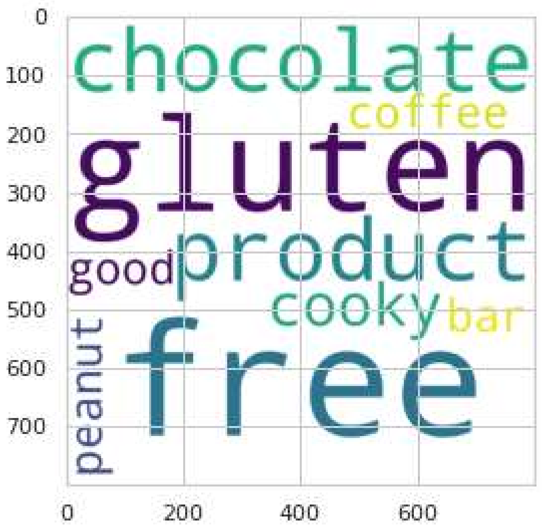


In [26]:

```
vineet = WordCloud(background_color="white", width=800, height=800, random_state=1).generate(Topics[8])
plt.imshow(vineet)
```

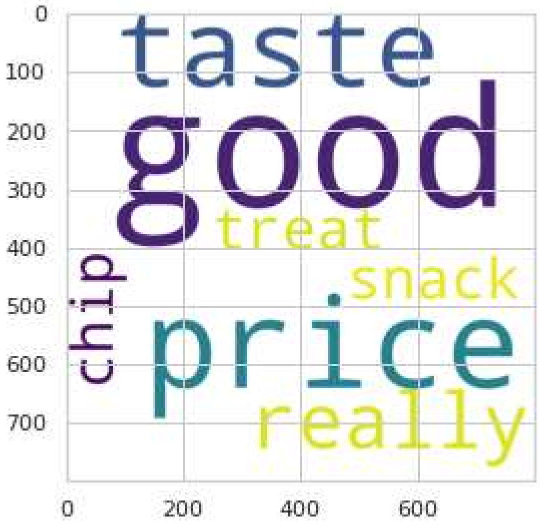
Out[26]:

```
<matplotlib.image.AxesImage at 0x7f8f25bf1950>
```



In [27]:
```
vineet = WordCloud(background_color="white", width=800, height=800, random_sta
te=1).generate(Topics[9])
plt.imshow(vineet)
```

Out[27]:
```
<matplotlib.image.AxesImage at 0x7f8f4329bc90>
```



THANK YOU