Datasets:

```
https://www.kaggle.com/bittlingmayer/amazonreviews
http://deepyeti.ucsd.edu/jianmo/amazon/index.html
```

In [ ]:

```python
import pandas as pd
# Reading the json dataset into a pandas dataframe
df = pd.read_json('E:/Internships/TCS-iON/Code/MyCode/Amazon/Prime_Pantry.json', orient = '
# Adding a column representing 1 for 'pos' and 0 for 'neg' sentiments
df['senti'] = df.apply(lambda x: 1 if x['overall'] >= 4 else 0, axis = 1)
# Deleting unnecessary columns
df = df.drop(['verified', 'reviewTime', 'asin', 'reviewerName', 'summary', 'unixReviewTime'
# Converting the data type to string
df['reviewText'] = df["reviewText"].astype("str")
# Converting all text to lowercase for use
df['reviewText'] = df['reviewText'].str.lower()
df.head()
```

overall	review	senti
5	good clinging	1
4	fantastic buy and a good plastic wrap. even t...	1
4	ok	1
3	saran cling plus is kind of like most of the c...	0
4	this is my go to plastic wrap so there isn't m...	1

saran cling plus is kind of like most of the cling wrap from glad. it is a very good quality plastic wrap, but the delivery system is poorly executed, and this makes usage more frustrating and less time-efficient. as convenience is one of the core selling points of this sort of product, how easy it is to use is just as important as how good the wrap itself is.

as another user here noted, getting this stuff out of the box and tearing off the portion you need to use is very difficult. substantial force is required to do this due to a cutting system that is lacking, and this can result in the box being damaged as it is not a very robust box. it can also cause the piece you are attempting to tear off fold up from the pressure upon being torn, leaving you with a tangled mess to untangle. as the blade itself does not do a great job cutting, sometimes the wrap will tear, leaving you with a piece that is a different size than you wanted. the physical location of where the wrap comes out relative to where it is cut at also results in one having to hold at an awkward angle to see what they are doing, making all of this more difficult.

i use clear wrap multiple times a day...the difference between a delivery system that is easy versus difficult means a lot of time savings, and a lot of frustration that does not have to occur. this is a good quality wrap, but like glad's wrap, it's delivery system is lacking compared to other products (such as the kirkland clear wraps) and so the convenience factor is reduced. so i think that you will find that an alternative clear wrap product with a better system for dispensing & cutting is more enjoyable to use, and much more convenient.

In [ ]:

```

```

In [ ]:

```
import re
import string
from nltk import WordNetLemmatizer
from nltk.stem.snowball import SnowballStemmer
from nltk.corpus import stopwords
# Initialising the nltk stop_words, stemmer and lemmatizer functions
stop_words = set(stopwords.words("english"))
lemmatizer = WordNetLemmatizer()
stemmer = SnowballStemmer("english")
# Creating a function for text cleaning
def textCleanser(myText):
    # Removing the name titles and the period symbols after it
    myText = re.sub(r'[mdsr]r(s)?\.', '', myText)
    # Removing punctuation
    myPunct = string.punctuation
    punctToSpace = str.maketrans(myPunct, len(myPunct)*' ')
    myText = myText.translate(punctToSpace)
    # Removing the '@username' mentions
    myText = re.sub(r'@\w+', '', myText)
    # Removing urls
    myText = re.sub(r'(http(s)?://)?(www\.)?.+\.com', '', myText)
    # Removing numbers
    myText = re.sub(r'\d+', '', myText)
    # Removing stopwords
    myText = [word for word in myText.split(' ') if not word in stop_words]
    myText = [word for word in myText if word != '']
    # Lemmatizing the text
    myText = [lemmatizer.lemmatize(token) for token in myText]
    # Stemming the text
    # myText = [stemmer.stem(token) for token in myText]
    return myText
for i in range(len(df['reviewText'])):
    df['reviewText'][i] = textCleanser(df['reviewText'][i])
df.head()
```

overall	review	senti
5	[good, clinging]	1
4	[fantastic, buy, good, plastic, wrap, even, th...	1
4	[ok]	1
3	[saran, cling, plus, kind, like, cling, wrap, ...	0
4	[go, plastic, wrap, much, bad, say, plastic, w...	1

'[saran', 'cling', 'plus', 'kind', 'like', 'cling', 'wrap', 'glad', 'good', 'quality', 'plastic', 'wrap', 'delivery', 'system', 'poorly', 'executed', 'make', 'usage', 'frustrating', 'le', 'time', 'efficient', 'convenience', 'one', 'core', 'selling', 'point', 'sort', 'product', 'easy', 'use', 'important', 'good', 'wrap', '\n\nas', 'another', 'user', 'noted', 'getting', 'stuff', 'box', 'tearing', 'portion', 'need', 'use', 'difficult', 'substantial', 'force', 'required', 'due', 'cutting', 'system', 'lacking', 'result', 'box', 'damaged', 'robust', 'box', 'also', 'cause', 'piece', 'attempting', 'tear', 'fold', 'pressure', 'upon', 'torn', 'leaving', 'tangled', 'mess', 'untangle', 'blade', 'great', 'job', 'cutting', 'sometimes', 'wrap', 'tear', 'leaving', 'piece', 'different', 'size', 'wanted', 'physical', 'location', 'wrap', 'come', 'relative', 'cut', 'also', 'result', 'one', 'hold', 'awkward', 'angle', 'see', 'making', 'difficult', '\n\ni', 'use', 'clear', 'wrap', 'multiple', 'time', 'day', 'difference', 'delivery', 'system', 'easy', 'versus', 'difficult', 'mean', 'lot', 'time', 'saving', 'lot', 'frustration', 'occur', 'good', 'quality', 'wrap', 'like', 'glad', 'wrap', 'delivery', 'system', 'lacking', 'compared', 'product', 'kirkland', 'clear', 'wrap', 'convenience', 'factor', 'reduced', 'think', 'find', 'alternative', 'clear', 'wrap', 'product', 'better', 'system', 'dispensing', 'cutting', 'enjoyable', 'use', 'much', 'convenient']"

In [ ]:

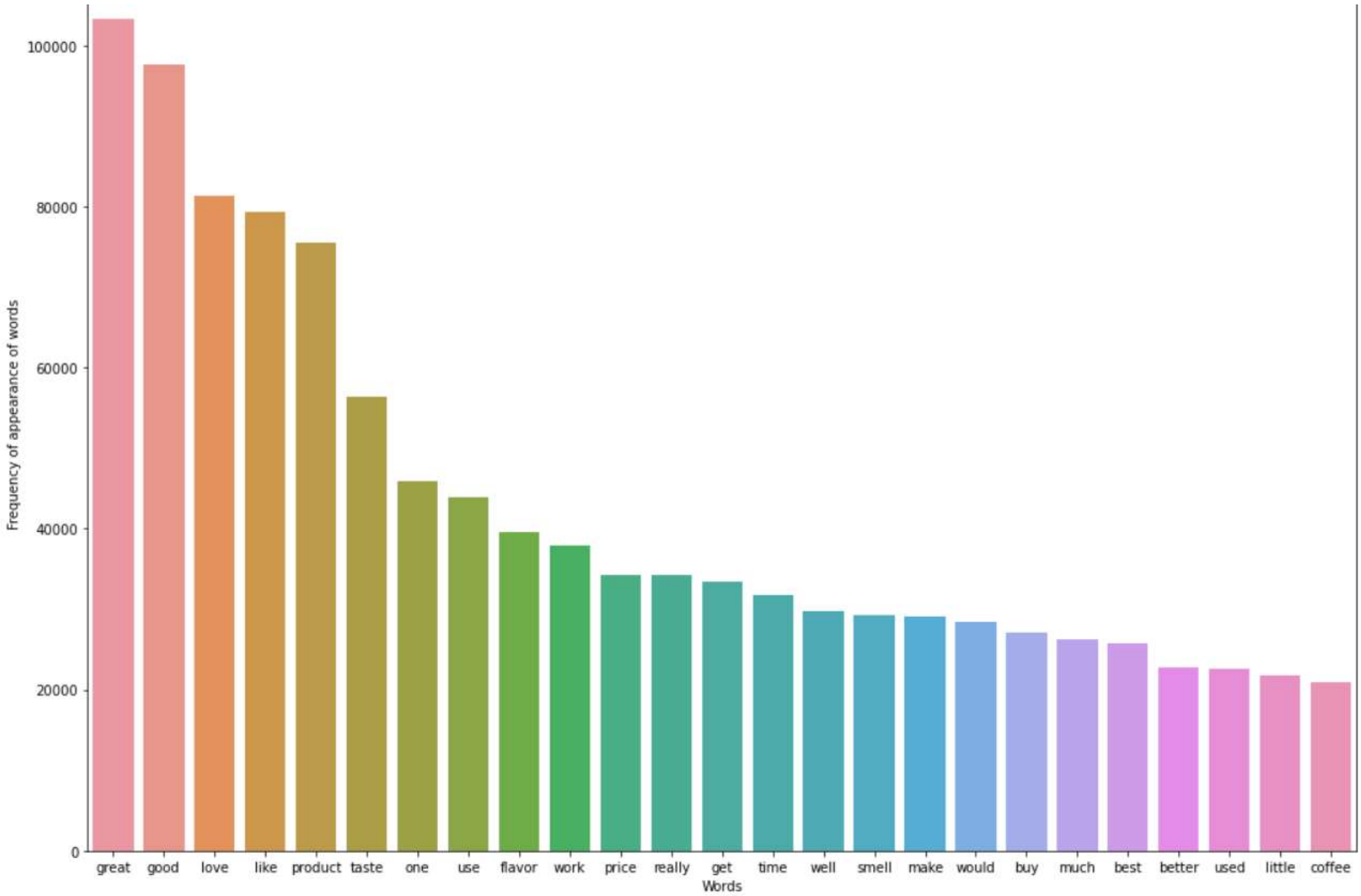
```
myReviews = []
for i in range(len(df['reviewText'])):
    for j in df['reviewText'][i]:
        if j != 'br' and j != 'http':
            myReviews.append(j)
```

In [ ]:

```
from collections import Counter
import collections
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.metrics import classification_report
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
import numpy as np
np.random.seed(1234)
# Initialising the Count Vectorizer
cv = CountVectorizer()
myBow = cv.fit_transform(myReviews)
wordFrequency = dict(zip(cv.get_feature_names(), np.asarray(myBow.sum(axis = 0)).ravel()))
wordCounter = collections.Counter(wordFrequency)
# Storing the frequency of appearance of words
dfWordCounter = pd.DataFrame(wordCounter.most_common(25), columns = ['word', 'frequency'])
```

In [ ]:

```
# Plotting the top 25 most frequently occuring words
plt.close('all')
fig, ax = plt.subplots(figsize = (17, 12))
sns.barplot(x = 'word', y = 'frequency', data = dfWordCounter, ax = ax)
sns.set_palette('pastel')
plt.xlabel('Words')
plt.ylabel('Frequency of appearance of words')
plt.show()
```


100000
80000
60000
40000
20000
0
Frequency of appearance of words
great good love like product taste one use flavor work price really get time well smell make would buy much best better used little coffee
Words

In [ ]:

In [ ]:

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense , Input , LSTM , Embedding, Dropout , Activation, GRU, Flatt
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.models import Model, Sequential
from keras.layers import Convolution1D
from keras import initializers, regularizers, constraints, optimizers, layers
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
# Splitting the dataframe to training and testing data
X_train, X_test, y_train, y_test = train_test_split(df["review"], df['senti'],test_size=0.2
# Initialising the tokenizer
max_features = 5000
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(X_train)
list_tokenized_train = tokenizer.texts_to_sequences(X_train)
maxlen = 130
X_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
y = y_train
embed_size = 128
# Initializing a bidirectional sequential LSTM using Adam optimizer, positive activation fu
model = Sequential()
model.add(Embedding(max_features, embed_size))
model.add(Bidirectional(LSTM(32, return_sequences = True)))
model.add(GlobalMaxPool1D())
model.add(Dense(20, activation="relu"))
model.add(Dropout(0.05))
model.add(Dense(1, activation="sigmoid"))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
batch_size = 100
epochs = 5
model.fit(X_t,y, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

In [ ]:

In [ ]:

```
# Testing the model
list_sentences_test = X_test
list_tokenized_test = tokenizer.texts_to_sequences(list_sentences_test)
X_te = pad_sequences(list_tokenized_test, maxlen=maxlen)
prediction = model.predict(X_te)
y_pred = (prediction > 0.5)
from sklearn.metrics import f1_score, confusion_matrix
print('F1-score: {0}'.format(f1_score(y_pred, y_test)))
print('Confusion matrix:')
confusion_matrix(y_pred, y_test)
```

In [ ]:

In [ ]:

```

```

F1-score:

```
0.9429220134457394
```

Confusion matrix:

```
[[10752, 3162 ],
 [ 5863, 74546]]
```

Accuracy:

```
90.43%
```