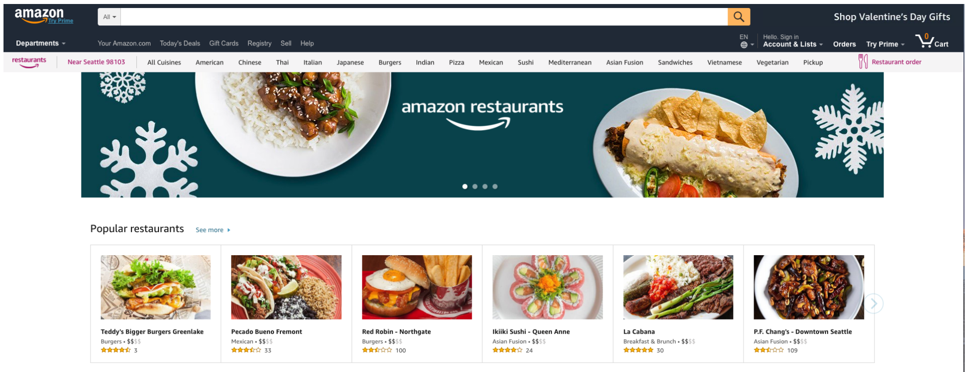


Introduction

Dive into the world of culinary exploration with our Amazon Food Review Dataset. This comprehensive collection captures the essence of diverse gastronomic experiences, offering insights into the myriad flavors and preferences of online consumers. As we sift through this data, anticipate a journey through taste, quality, and consumer satisfaction. From trending products to hidden gems, our dataset unravels the tapestry of Amazon's vast food offerings. Whether you're a researcher, marketer, or simply a food enthusiast, this review compilation provides a valuable resource to understand and analyze the dynamic landscape of online food reviews on Amazon in a concise and informative manner.

In [170…

```
# # Amazon_food_Review dataset
```

1 import Necessary Library

In [171…

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from wordcloud import WordCloud

import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer

from collections import Counter
from numpy import where

from imblearn.over_sampling import SMOTE
from sklearn.decomposition import PCA

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn import metrics
```

```
from scipy.sparse import hstack, vstack

from prettytable import PrettyTable
from scipy.stats import loguniform # Log-uniform is useful for searching pend
from sklearn.model_selection import RepeatedStratifiedKFold, RandomizedSearch
```

2 import Dataset

In [172…

```
nltk.download('stopwords')
```

```
[nltk_data] Error loading stopwords: <urlopen error [Errno -3]
[nltk_data]     Temporary failure in name resolution>
```

Out[172…

```
False
```

In [173…

```
nltk.download('punkt')
```

```
[nltk_data] Error loading punkt: <urlopen error [Errno -3] Temporary
[nltk_data]     failure in name resolution>
```

Out[173…

```
False
```

In [174…

```
df = pd.read_csv("/kaggle/input/amazon-fine-food-reviews/Reviews.csv")
```

3 Data Analysis

In [175…

```
df.head()
```

Out[175…

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
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 [176…

```
df.tail()
```

Out[176…

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator
568449	568450	B001EO7N10	A28KG5XORO54AY	Lettie D. Carter	0
568450	568451	B003S1WTCU	A3I8AFVPEE8KI5	R. Sawyer	0
568451	568452	B004I613EE	A121AA1GQV751Z	pksd "pk_007"	2
568452	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	1
568453	568454	B001LR2CU2	A3LGQPJCZVL9UC	srfell17	0

In [177…

```
df.shape
```

Out[177…

```
(568454, 10)
```

In [178…

```
df.columns
```

Out[178…

```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

In [179…

```
df["Score"].value_counts()
```

Out[179…

```
Score
5    363122
4     80655
1     52268
3     42640
2     29769
```

```
Name: count, dtype: int64
```

4 Data cleaning and Preprocessing

In [180…

```
# Limiting current dataset to 5000 rows
df = df[:10000]
```

In [181…

```
print('No. of datapoints/rows: {}'.format(df.shape[0]))
print('No. of features/columns: {}'.format(df.shape[1]))
```

```
No. of datapoints/rows: 10000
No. of features/columns: 10
```

In [182…

```
print("Feature names: \n{}".format(df.columns))
```

```
Feature names:
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

- **Id**: Just the Row Number
- **ProductId**: Unique identifier for the product
- **UserId**: Unqiue identifier for the user
- **ProfileName**: Profile name of the user
- **HelpfulnessNumerator**: Number of users who found the review helpful
- **HelpfulnessDenominator**: Number of users who indicated whether they found the review helpful or not
- **Score**: Rating between 1 and 5
- **Time**: Timestamp for the review
- **Summary**: Brief summary of the review
- **Text**: Text of the review

In [183…

```
df.isna().sum()
```

Out[183…

```
Id                        0
ProductId                 0
UserId                    0
ProfileName               0
HelpfulnessNumerator      0
HelpfulnessDenominator    0
Score                     0
Time                      0
Summary                   0
Text                      0
dtype: int64
```

In [184…

```
df.isnull().sum()
```

Out[184…

```
Id                        0
ProductId                 0
UserId                    0
ProfileName               0
HelpfulnessNumerator      0
HelpfulnessDenominator    0
Score                     0
```

```
Time                      0
Summary                   0
Text                      0
dtype: int64
```

Inference:

Only `ProfileName` and `Summary` are Null or Missing, so we can continue without removing those rows as `UserId` and `Text` are present and we can use these features

In [185…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Id                      10000 non-null  int64
 1   ProductId               10000 non-null  object
 2   UserId                  10000 non-null  object
 3   ProfileName             10000 non-null  object
 4   HelpfulnessNumerator    10000 non-null  int64
 5   HelpfulnessDenominator  10000 non-null  int64
 6   Score                   10000 non-null  int64
 7   Time                    10000 non-null  int64
 8   Summary                 10000 non-null  object
 9   Text                    10000 non-null  object
dtypes: int64(5), object(5)
memory usage: 781.4+ KB
```

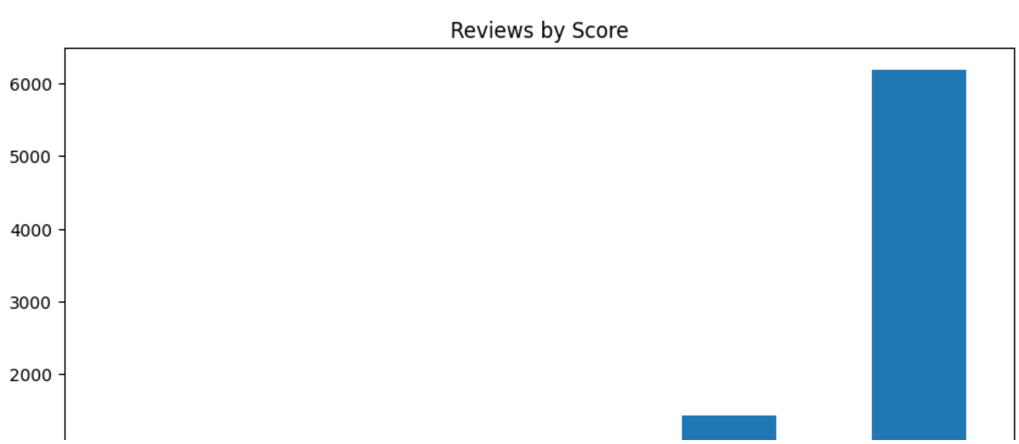
5| Data visualisation 📊 📉

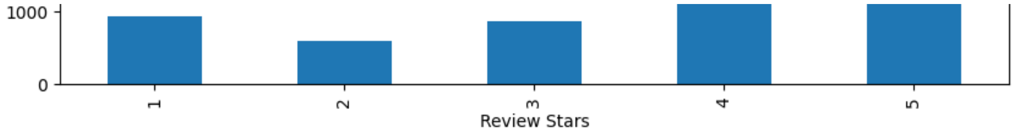
EDA (Exploratory Data Analysis)

5.1 Bar Plot

In [186…

```
ax = df['Score'].value_counts().sort_index().plot(kind='bar', title='Reviews
ax.set_xlabel('Review Stars')
plt.show()
```





In [187...

```
df['Score'].value_counts()
```

Out[187...

```
Score
5    6183
4    1433
1     932
3     862
2     590
Name: count, dtype: int64
```

Check duplicate values

In [188...

```
duplicates = df[df.duplicated(subset=['ProductId', 'UserId', 'ProfileName', '
duplicates
```

Out[188...

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	He
466	467	B000G6RYNE	A3PJZ8TU8FDQ1K	Jared Castle	0	
574	575	B000G6RYNE	A3PJZ8TU8FDQ1K	Jared Castle	2	
2334	2335	B0001FQVCK	A5D06XJHDXK75	C. Po	3	
2336	2337	B0001FQVCK	A5D06XJHDXK75	C. Po	1	
2613	2614	B0016FY6H6	A3I4PCBRENJNG2	L. Cain	4	

2636	2637	B0016FY6H6	A2NLZ3M0OJV9NX	Mark Bodzin	3
2647	2648	B0016FY6H6	A2NLZ3M0OJV9NX	Mark Bodzin	0
2653	2654	B0016FY6H6	A3I4PCBRENJNG2	L. Cain	0
2941	2942	B0002TJAZK	A3TVZM3ZIXG8YW	christopher hayes	7
2943	2944	B0002TJAZK	A2ISKAWUPGGOLZ	M. S. Handley	2
2946	2947	B0002TJAZK	A2ISKAWUPGGOLZ	M. S. Handley	0
2947	2948	B0002TJAZK	A3TVZM3ZIXG8YW	christopher hayes	0
5934	5935	B001O2IX8E	A3KDZCQ82JFWLN	Phoebe Oh	2

5958	5959	B001O2IX8E	A3KDZCQ82JFWLN	Phoebe Oh	0
6516	6517	B005O8BLLU	APH7I7OZ8WUJP	J. Simpson	0
6517	6518	B005O8BLLU	APH7I7OZ8WUJP	J. Simpson	0
8522	8523	B003VXFK44	A10H24TDLK2VDP	William Jens Jensen	0
8523	8524	B003VXFK44	A10H24TDLK2VDP	William Jens Jensen	0
8702	8703	B003VXFK44	A10H24TDLK2VDP	William Jens Jensen	2
9231	9232	B006N3IG4K	A10H24TDLK2VDP	William Jens Jensen	0
9232	9233	B006N3IG4K	A10H24TDLK2VDP	William Jens Jensen	0

9411	9412	B006N3IG4K	A10H24TDLK2VDP	William Jens Jensen	2

In [189…

```
df.duplicated(subset=['ProductId', 'UserId', 'ProfileName', 'Time', 'Text'],
```

Out[189…

```
False    9978
True       22
Name: count, dtype: int64
```

In [190…

```
duplicates[duplicates['ProductId']=='B0016FY6H6']
```

Out[190…

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Hel
2613	2614	B0016FY6H6	A3I4PCBRENJNG2	L. Cain	4	
2636	2637	B0016FY6H6	A2NLZ3M0OJV9NX	Mark Bodzin	3	
2647	2648	B0016FY6H6	A2NLZ3M0OJV9NX	Mark Bodzin	0	
2653	2654	B0016FY6H6	A3I4PCBRENJNG2	L. Cain	0	

Data Cleaning (2nd part)

Drop duplicates

In [191…

```
# Check original shape of the dataset
df.shape
```

Out[191…

```
(10000, 10)
```

In [192…

```
# Drop the duplicates
df.drop_duplicates(subset=['ProductId', 'UserId', 'ProfileName', 'Time', 'Tex
df.shape
```

Out[192…

```
(9988, 10)
```

Helpfulness numerator should not exceed Helpfulness denominator

In [193…

```
df[df["HelpfulnessNumerator"] > df["HelpfulnessDenominator"]]
```

Out[193…

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominato

In [194…

```
print(f"No. of Datapoints BEFORE discarding : {df.shape[0]}")

df = df[df["HelpfulnessNumerator"] <= df["HelpfulnessDenominator"]]

print(f"No. of Datapoints AFTER discarding : {df.shape[0]}")
```

```
No. of Datapoints BEFORE discarding : 9988
No. of Datapoints AFTER discarding : 9988
```

6 Feature Engineering

In [195…

```
print("Positive reviews:", df[df['Score']>3].shape[0])
print("Negative reviews:", df[df['Score']<=3].shape[0])
```

```
Positive reviews: 7612
Negative reviews: 2376
```

In [196…

```
df['Review'] = [1 if x>3 else 0 for x in df['Score']] # set 1 for positive re
df.head(5)
```

Out[196…

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
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 [197…

```
# Check negative and positive reviews (1, 2, 3 - negative; 4 and 5 - positive
print("Negative values with scores 1, 2 and 3:", len(df[df['Review']==0]))
print("Positive values with score 4 and 5:", len(df[df['Review']==1]))
```

```
Negative values with scores 1, 2 and 3: 2376
Positive values with score 4 and 5: 7612
```

Add *Word Count* feature

In [198…

```
df['WordCount'] = df['Text'].apply(lambda x: len(x.split()))
df.head()
```

Out[198…

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres	1	

2	3	B000LQOCH0	ABXLMWJIXXAIN	"Natalia Corres"	1
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0

Add *Character Count* feature

In [199…

```
df['CharacterCount'] = df['Text'].apply(lambda x: len(x))
df.head()
```

Out[199…

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
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	

Add *Helpfulness percentage* feature

Add Helpfulness percentage feature

In [200…

```
df["HelpfulnessPercentage"] = df[["HelpfulnessNumerator","HelpfulnessDenomina
df.head(-5)
```

Out[200…

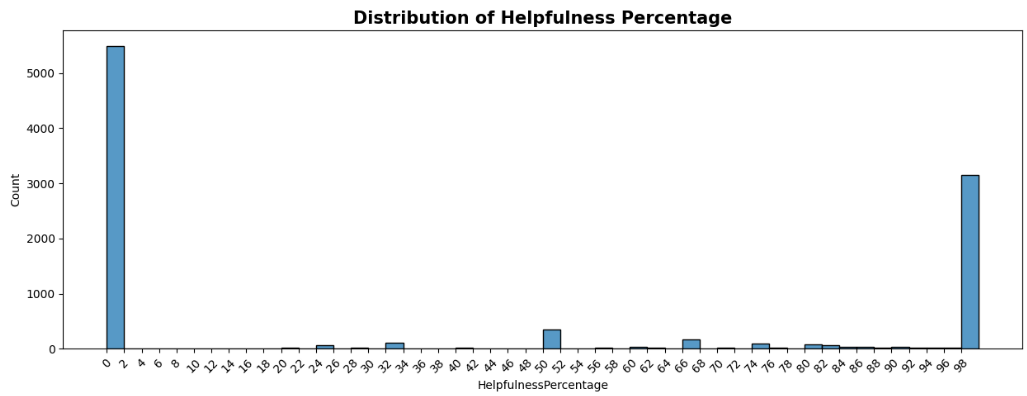
	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	H
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	
...	...	...	...	...	...	
9990	9991	B000P41A28	A82CL6H9NWSJC	Carl Nothnagel	6	
9991	9992	B000P41A28	A181WVPZSOKTVV	GRIZZLY	12	
9992	9993	B000P41A28	A3HINZRNCW1SKA	Happy Mom	1	

9993	9994	B000P41A28	AV3IMDC3C0F8	Miss K	1
9994	9995	B000P41A28	A350OL4V8DV5YK	Helen Avramenko	3

9983 rows × 14 columns

In [201…

```
# Check the distribution of helpfulness percentage
plt.figure(figsize=(15,5))
sns.histplot(data=df["HelpfulnessPercentage"], bins=50)
plt.title("Distribution of Helpfulness Percentage",fontweight='bold', fontsiz
plt.xticks(range(0,100,2), rotation=45)
plt.show()
```



Adding *Helpfulness Indicator* Feature

In [202…

```
df.loc[df["HelpfulnessPercentage"] >= 75, 'HelpfulnessIndicator'] = 'Useful'
df.loc[(df["HelpfulnessPercentage"] > 40) & (df["HelpfulnessPercentage"] < 75
df.loc[(df["HelpfulnessPercentage"] > 0) & (df["HelpfulnessPercentage"] <= 40
df.loc[df["HelpfulnessPercentage"] == 0, 'HelpfulnessIndicator'] = 'Not Avail
df.head()
```

Out[202…

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	

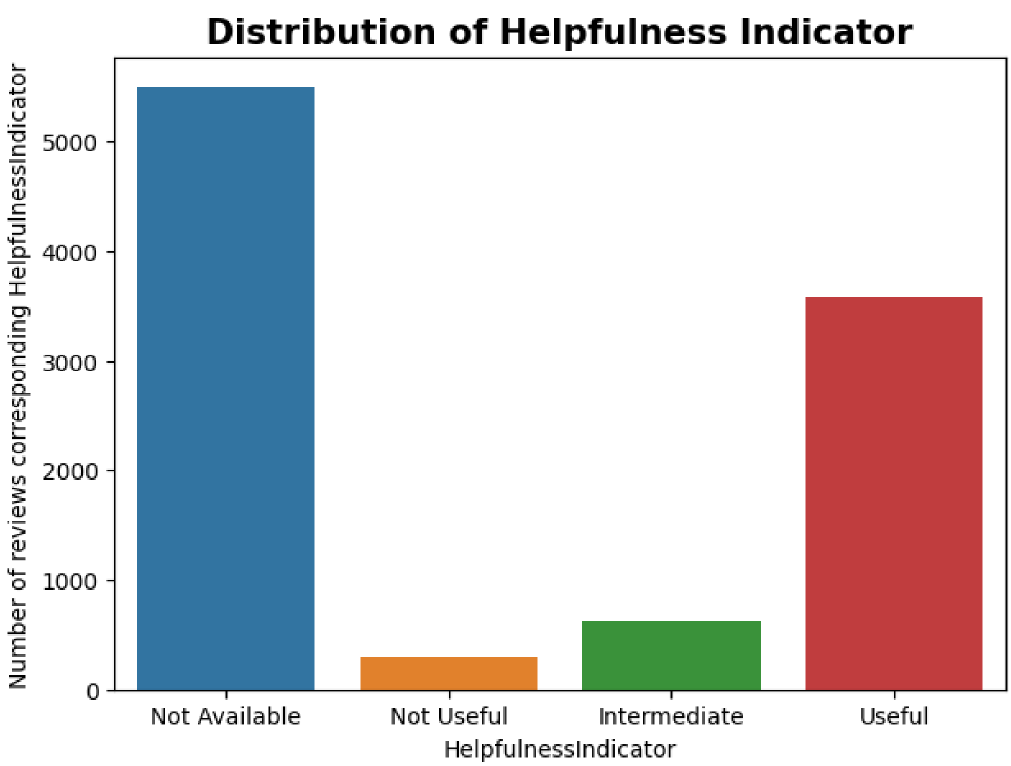
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0

In [203…

```
plt.figure(figsize=(7,5))
sns.countplot(df, x='HelpfulnessIndicator', order=["Not Available","Not Usefu
plt.title("Distribution of Helpfulness Indicator",fontweight='bold', fontsize
plt.xlabel("HelpfulnessIndicator")
plt.ylabel("Number of reviews corresponding HelpfulnessIndicator")
plt.show()

print()

print(df['HelpfulnessIndicator'].value_counts()[[0,3,2,1]])
```

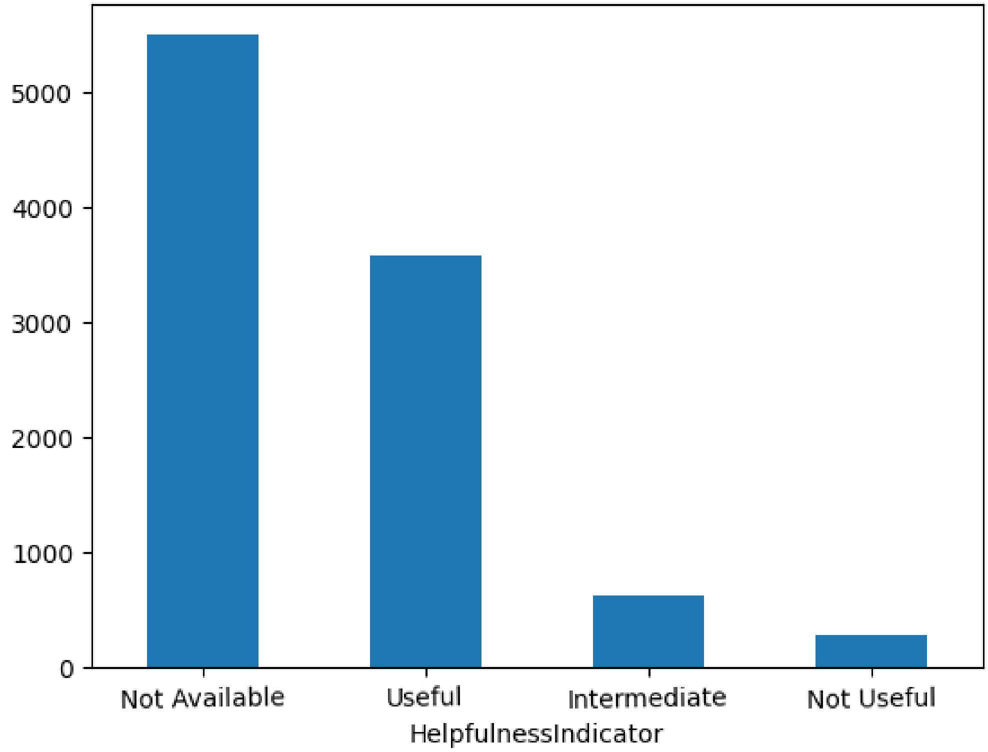



```
HelpfulnessIndicator
Not Available    5494
Not Useful        290
Intermediate      629
Useful           3575
Name: count, dtype: int64
```

```
Name: count, dtype: int64
```

In [204…

```
df.HelpfulnessIndicator.value_counts().plot(kind='bar', rot=1.0)
plt.show()
print("\nCount of Usefulness of Reviews:")
print(df.HelpfulnessIndicator.value_counts())
```




```
Count of Usefulness of Reviews:
HelpfulnessIndicator
Not Available    5494
Useful           3575
Intermediate      629
Not Useful        290
Name: count, dtype: int64
```

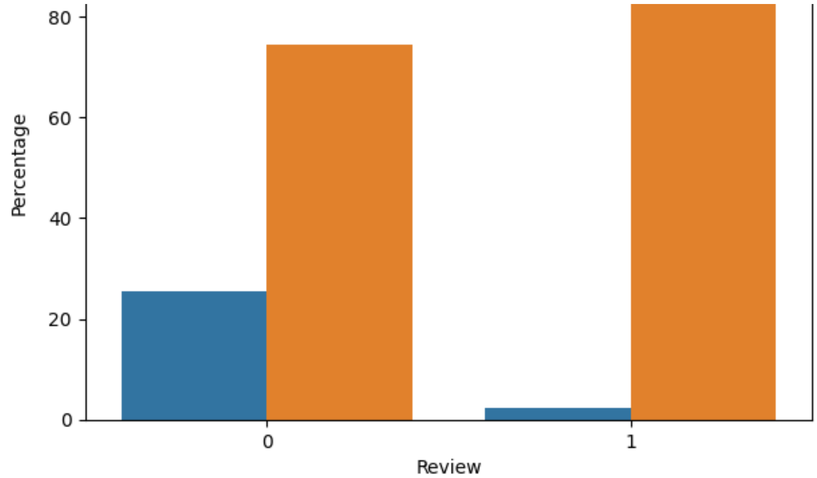
EDA 2

Distribution of useful and non-useful reviews in each of the set of Positive and Negative Reviews

In [205…

```
df_temp = df[(df["HelpfulnessIndicator"]!= "Not Available") & (df["Helpfulnes
df_temp_1 = df_temp["HelpfulnessIndicator"].groupby(df_temp["Review"]).value_
df_temp_1 = df_temp_1*100
df_temp_1 = df_temp_1.rename("Percentage").reset_index()

plt.figure(figsize=(7,5))
sns.barplot(data=df_temp_1, x="Review", y="Percentage", hue="HelpfulnessIndic
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.);
plt.show()
print()
df_temp_1
```



Out[205…

	Review	HelpfulnessIndicator	Percentage
0	0	Useful	74.440518
1	0	Not Useful	25.559482
2	1	Useful	97.579576
3	1	Not Useful	2.420424

Inference:

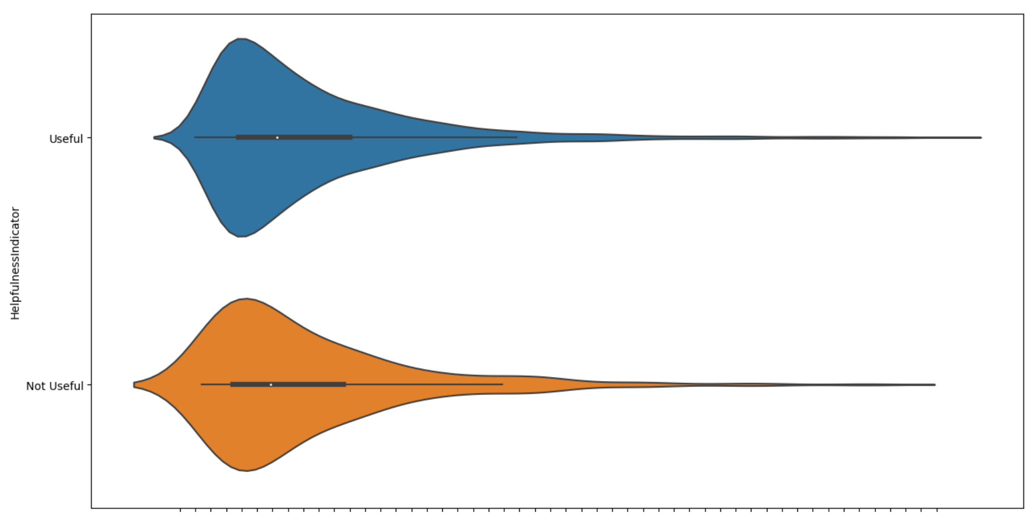
- People find both positive and negative reviews useful
- It's very rare that positive reviews are not useful, meaning the reviews are well written and true in the dataset

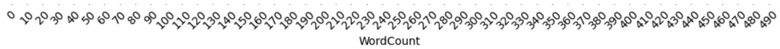
Usefulness vs Length of the Review

In [206…

```
# Consider reviews with 500 words or less
temp_df_useful_nonuseful_500wc = df[(df["HelpfulnessIndicator"]!= "Not Availa
plt.figure(figsize=(15,8))
sns.violinplot(x='WordCount', y='HelpfulnessIndicator', data=temp_df_useful_r
plt.xticks(range(0,500,10), rotation=45)
plt.show()
print()

temp_df_useful_nonuseful_500wc["WordCount"].groupby(temp_df_useful_nonuseful_
```





Out[206…

	count	mean	std	min	25%	50%	75%	max
HelpfulnessIndicator								
Not Useful	287.0	81.975610	68.004238	14.0	34.5	59.0	106.0	445.0
Useful	3558.0	84.632659	68.806229	10.0	38.0	63.0	110.0	492.0

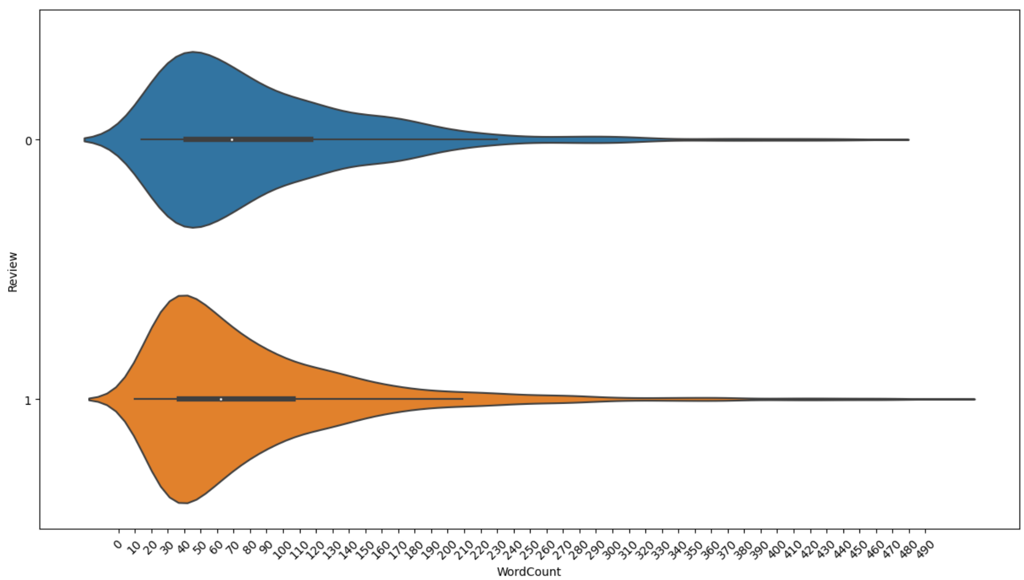
Inference:

- Useful reviews are concise.
- Not useful reviews are lengthy

Review Length vs Negative/Positive

In [207…

```
plt.figure(figsize=(15,8))
sns.violinplot(x='WordCount', y='Review', data=temp_df_useful_nonuseful_500wc
plt.xticks(range(0,500,10), rotation=45)
plt.show()
print()
temp_df_useful_nonuseful_500wc["WordCount"].groupby(temp_df_useful_nonuseful_
```



Out[207…

	count	mean	std	min	25%	50%	75%	max
Review								
0	843.0	88.946619	66.856483	14.0	41.0	69.0	117.0	445.0
1	3002.0	83.167222	69.219604	10.0	37.0	62.0	106.0	492.0

Inference:

- Negative reviews are lengthy

Data Processing 1

In [208…

```
import re
def clean_text(reviews_df):
  cleaned_reviews_df = []
  cleaned_reviews = ""
  for text in reviews_df:
    text = text.lower() # Converting to lowercase
    pattern = re.compile('<.*?>')
    text = re.sub(pattern, ' ', text) # Removing HTML tags
    text = re.sub(r'[?|!|\'|"|#]', r'', text)
    text = re.sub(r'[.|,|)|(|\|/]', r' ', text) # Removing Punctuations
    words = [word for word in text.split() if word not in stopwords.words('er
    cleaned_reviews_df.append(words)
    cleaned_reviews = list(map(' '.join, cleaned_reviews_df))
  return cleaned_reviews
```

In [209…

```
df['CleanedText'] = clean_text(df['Text'])
df['CleanedText'][56:90]
```

Out[209…

```
56    deal awesome arrived halloween indicated enoug...
57    chocolate say great variety everything family ...
58    great product nice combination chocolates perf...
59    halloween sent bag daughters class share choco...
60    watch prices assortment good get gold box purc...
61    bag candy online pretty expensive cheaper orde...
62                arrived 6 days stale could eat 6 bags
63    used endurolyte product several years pill pow...
64    product serves well source electrolytes long r...
65    stuff really works preventing cramping middle ...
66    us low carb diet little tablets thing two year...
67    purchased mango flavor doesnt take like mango ...
68    youre impulsive like $6 ok dont get wrong qual...
69    sooooo deliscious bad ate em fast gained 2 pds...
70    albanese gummi bears rings good tasty high qua...
71    grape gummy bears hard find area fact pretty m...
72    ordered two two raspberry latice tarts directl...
73    buyer beware please sweetener everybody maltit...
74                                 okay would go way buy
75    tea flavor whole brunch artifial flavors retur...
76    looked like perfect snack trail mix unfortunat...
77    taste really good purchasing different brand s...
78    taste great berries melted may order winter or...
79    know cannot make tea good granted south know n...
80    peppermint stick delicious fun eat dad got one...
81    great gift ages purchased giant canes recipien...
82    know product title says molecular gastronomy d...
83    dogs like flavors tried dog food reason itchin...
84    awesome dog food however given boston severe r...
85    three dogs love food bought specifically one d...
86    dog ton allergies environmental food prescript...
87    shepherd collie mix ibs vet recommended limite...
88    natural balance dry dog food lamb meal brown r...
89    great food love idea one food ages & breeds ît...
Name: CleanedText, dtype: object
```

In [210…

```
df.head()
```

Out[210…

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
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

EDA 3 - Word Cloud

Word cloud for all reviews

In [211…

```
all_text = " ".join(review for review in df['CleanedText'])

wordcloud = WordCloud(stopwords=stopwords.words('english'), background_color=
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Word cloud for positive review

In [212…

```
positive_text = " ".join(review for review in df['CleanedText'][df['Review']=

wordcloud = WordCloud(stopwords=stopwords.words('english'), background_color=
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Word cloud for negative reviews

In [213…

```
negative_text = " ".join(review for review in df['CleanedText'][df['Review']=

wordcloud = WordCloud(stopwords=stopwords.words('english'), background_color=
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Inference:

There will be group of words that indicate negative review, so would need to apply n-gram on that

Finding most common words in NEGATIVE REVIEWS and then plotting the word cloud:

In [214…

```
# Tokenize the sentences in the corpus and create a dictionary with sentences
wordfreq = {}
tokens = nltk.word_tokenize(negative_text)
for t in tokens:
    if t not in wordfreq.keys():
        wordfreq[t] = 1
    else:
        wordfreq[t] += 1
# print(wordfreq)
```

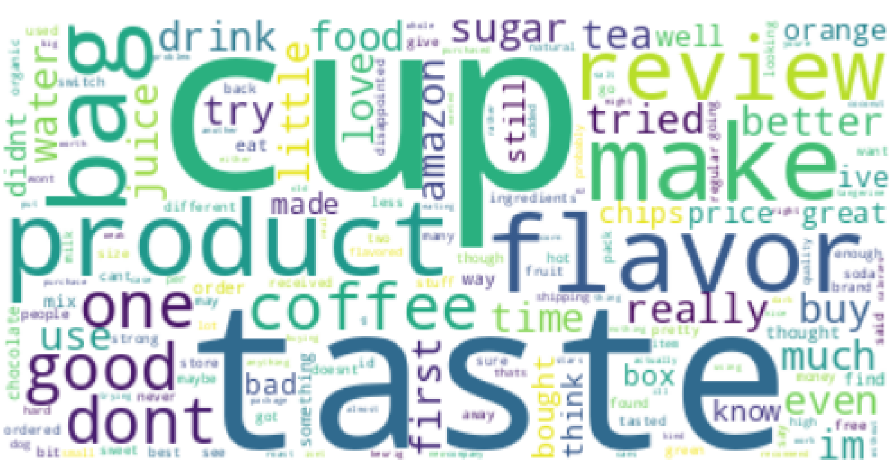
In [215…

```
# Filter down to 200 most frequently ocurring words:
import heapq
most_freq = heapq.nlargest(200, wordfreq, key=wordfreq.get)
print(most_freq)
```

```
['like', 'coffee', 'taste', 'product', 'one', 'would', 'good', 'flavor', 'don
t', 'really', 'much', 'even', 'get', 'buy', 'food', 'better', 'im', 'drink', 's
ugar', 'tried', 'water', ':', '-', 'also', 'amazon', 'juice', 'little', 'time',
'tea', 'cup', 'first', 'try', 'use', '$', 'love', 'price', 'bad', 'box', 'grea
t', 'make', '3', 'bought', 'orange', 'chips', 'well', 'made', 'didnt', 'know',
'still', 'think', 'ive', '2', 'thought', 'way', 'eat', 'something', '%', 'mix',
'chocolate', 'could', 'order', 'find', 'ingredients', 'tastes', 'want', 'howeve
r', 'sweet', 'used', 'say', 'bag', 'found', 'got', 'two', 'hot', 'soda', 'doesn
t', 'brand', 'bit', 'flavors', 'different', 'since', 'disappointed', 'ordered',
'products', 'tasted', 'less', 'never', 'best', 'go', 'though', '1', '4', 'giv
e', 'money', 'sure', 'many', 'cant', 'organic', 'going', ';', 'milk', '--', 'aw
ay', 'pack', 'may', 'back', 'store', 'size', 'green', 'switch', 'reviews', '5',
'said', 'looking', 'see', 'stuff', 'people', 'received', 'id', 'regular', 'fre
e', 'enough', 'strong', 'maybe', 'pretty', 'fruit', 'shipping', 'natural', 'ite
m', 'high', 'flavored', 'wont', 'hard', 'thats', 'per', 'added', 'probably', 'd
og', 'small', 'quality', 'lot', 'cups', 'purchase', 'another', 'put', 'anythin
g', 'isnt', '&', 'k-cups', 'keurig', 'work', 'right', 'package', 'thing', 'ol
d', 'actually', 'purchased', 'almost', 'buying', 'might', 'using', 'real', 'bi
g', 'bags', 'ok', '100', '8', 'review', 'ill', 'recommend', 'oz', 'youre', 'eve
r', 'whole', 'kind', 'stars', 'salt', 'tangerine', 'new', 'roast', 'calories',
'corn', 'nothing', 'case', 'dark', 'cans', '12', 'problem', 'weak', 'eating',
'without', 'coconut', 'either', 'makes', 'company', 'nice', 'worth', 'rather',
'trying', 'wanted']
```

In [216…

```
top_200_words = " ".join(word for word in most_freq)
wordcloud_top_200 = WordCloud(background_color="white").generate(top_200_word
plt.imshow(wordcloud_top_200, interpolation='bilinear')
plt.axis("off")
plt.show()
```



Data Processing 2

Stemming

In [217…

```
snow = nltk.stem.SnowballStemmer('english')
final_X = []
for text in df['CleanedText']:
  words = [snow.stem(word) for word in text.split()]
  final_X.append(words)
final_X[:10]
```

Out[217…

```
[['bought',
  'sever',
  'vital',
  'can',
  'dog',
  'food',
  'product',
  'found',
  'good',
  'qualiti',
  'product',
  'look',
  'like',
  'stew',
  'process',
  'meat',
  'smell',
  'better',
  'labrador',
  'finicki',
  'appreci',
  'product',
  'better'],
 ['product',
  'arriv',
  'label',
  'jumbo',
  'salt',
  'peanut',
  'peanut',
  'actual',
  'small',
  'size',
  'unsalt',
  'sure',
  'error',
  'vendor',
  'intend',
  'repres',
  'product',
  'jumbo'],
 ['confect',
  'around',
  'centuri',
  'light',
  'pillowi',
  'citrus',
  'gelatin',
  'nut',
  '-',
  'case',
  'filbert',
  'cut',
```

```
  'tini',
  'squar',
  'liber',
  'coat',
  'powder',
  'sugar',
  'tini',
  'mouth',
  'heaven',
  'chewi',
  'flavor',
  'high',
  'recommend',
  'yummi',
  'treat',
  'familiar',
  'stori',
  'c',
  'lewi',
  'lion',
  'witch',
  'wardrob',
  '-',
  'treat',
  'seduc',
  'edmund',
  'sell',
  'brother',
  'sister',
  'witch'],
 ['look',
  'secret',
  'ingredi',
  'robitussin',
  'believ',
  'found',
  'got',
  'addit',
  'root',
  'beer',
  'extract',
  'order',
  'good',
  'made',
  'cherri',
  'soda',
  'flavor',
  'medicin'],
 ['great',
  'taffi',
  'great',
  'price',
  'wide',
  'assort',
  'yummi',
  'taffi',
  'deliveri',
  'quick',
  'taffi',
  'lover',
  'deal'],
 ['got',
  'wild',
  'hair',
  'taffi',
  'order',
  'five',
```

```
  'pound',
  'bag',
  'taffi',
  'enjoy',
  'mani',
  'flavors:',
  'watermelon',
  'root',
  'beer',
  'melon',
  'peppermint',
  'grape',
  'etc',
  'complaint',
  'bit',
  'much',
  'red',
  'black',
  'licorice-flavor',
  'piec',
  'particular',
  'favorit',
  'kid',
  'husband',
  'last',
  'two',
  'week',
  'would',
  'recommend',
  'brand',
  'taffi',
  '--',
  'delight',
  'treat'],
 ['saltwat',
  'taffi',
  'great',
  'flavor',
  'soft',
  'chewi',
  'candi',
  'individu',
  'wrap',
  'well',
  'none',
  'candi',
  'stuck',
  'togeth',
  'happen',
  'expens',
  'version',
  'fraling',
  'would',
  'high',
  'recommend',
  'candi',
  'serv',
  'beach-them',
  'parti',
  'everyon',
  'love'],
 ['taffi',
  'good',
  'soft',
  'chewi',
  'flavor',
  'amaz',
```

```
  'would',
  'definit',
  'recommend',
  'buy',
  'satisfi'],
 ['right',
  'im',
  'most',
  'sprout',
  'cat',
  'eat',
  'grass',
  'love',
  'rotat',
  'around',
  'wheatgrass',
  'rye'],
 ['healthi',
  'dog',
  'food',
  'good',
  'digest',
  'also',
  'good',
  'small',
  'puppi',
  'dog',
  'eat',
  'requir',
  'amount',
  'everi',
  'feed']]
```

In [218…

```
final_y = df['Review']
```

Convert to bag of words

In [219…

```
stemmed_X = []
for row in final_X:
    sentence = ''
    for word in row:
        sentence = sentence + ' ' + word
    stemmed_X.append(sentence.strip())
```

In [220…

```
stemmed_X[:5]
```

Out[220…

```
['bought sever vital can dog food product found good qualiti product look lik
e stew process meat smell better labrador finicki appreci product better',
 'product arriv label jumbo salt peanut peanut actual small size unsalt sure
error vendor intend repres product jumbo',
 'confect around centuri light pillowi citrus gelatin nut - case filbert cut
tini squar liber coat powder sugar tini mouth heaven chewi flavor high recomm
end yummi treat familiar stori c lewi lion witch wardrob - treat seduc edmund
sell brother sister witch',
 'look secret ingredi robitussin believ found got addit root beer extract ord
er good made cherri soda flavor medicin',
 'great taffi great price wide assort yummi taffi deliveri quick taffi lover
deal']
```

In [221…

```
count_vect = CountVectorizer(max_features=100)
bow_X = count_vect.fit_transform(stemmed_X)
final_X = bow_X
```

```
print(final_X[:5])
```

```
(0, 7)        1
(0, 22)       1
(0, 32)       1
(0, 68)       3
(0, 33)       1
(0, 38)       1
(0, 50)       1
(0, 48)       1
(0, 5)        2
(1, 68)       2
(2, 81)       1
(2, 31)       1
(2, 41)       1
(2, 71)       1
(2, 88)       2
(3, 33)       1
(3, 38)       1
(3, 50)       1
(3, 31)       1
(3, 44)       1
(3, 39)       1
(3, 62)       1
(3, 53)       1
(4, 40)       2
(4, 67)       1
```

In [222…

```
print("Count of final_X:")
print(final_X.shape[0])
print()
print("Count of final_y:")
print(final_y.value_counts())
```

```
Count of final_X:
9988

Count of final_y:
Review
1    7612
0    2376
Name: count, dtype: int64
```

Plot the bag of words (before balancing)

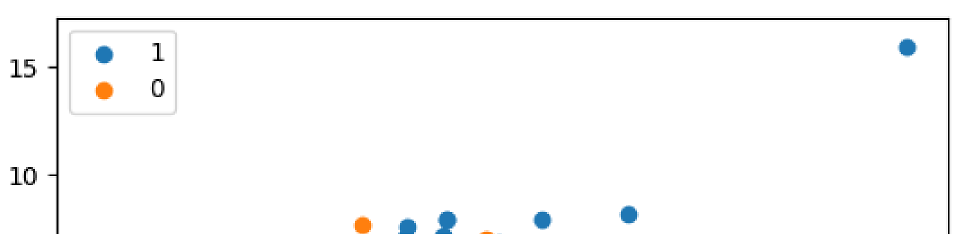
In [223…

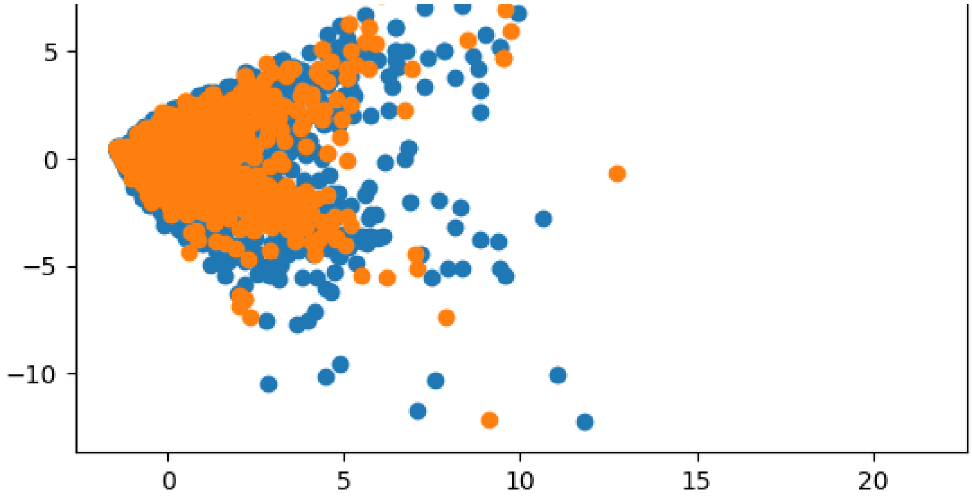
```
pca = PCA(n_components = 2)
```

In [224…

```
PCA_X = pca.fit_transform(final_X.toarray()) # Apply PCA to plot 2 dimensions

counter = Counter(final_y)
for label, _ in counter.items():
 row_ix = where(final_y == label)[0]
 plt.scatter(PCA_X[row_ix, 0], PCA_X[row_ix, 1], label=str(label))
plt.legend()
plt.show()
```





7 | Applying SMOTE 'Balance Dataset'

In [225…

```
oversample = SMOTE()

X_resampled, y_resampled = oversample.fit_resample(final_X, final_y)
X_resampled.shape
```

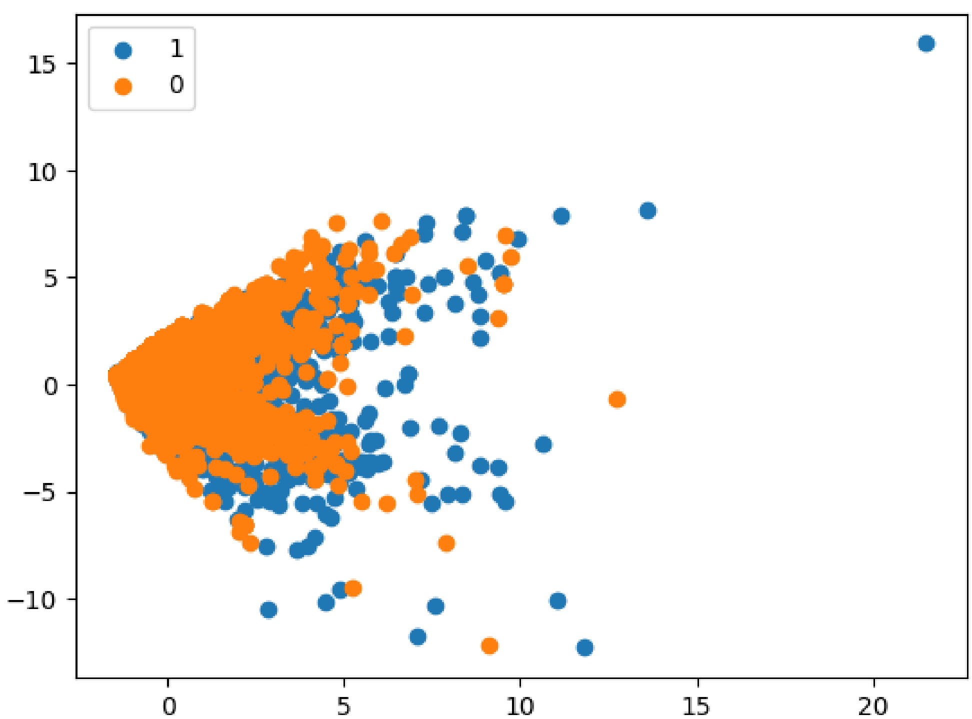
Out[225…

```
(15224, 100)
```

In [226…

```
PCA_SMOTE_X = pca.transform(X_resampled.toarray())

for label, _ in counter.items():
 row_ix = where(y_resampled == label)[0]
 plt.scatter(PCA_SMOTE_X[row_ix, 0], PCA_SMOTE_X[row_ix, 1], label=str(label)
plt.legend()
plt.show()
```



In [227…

```
print("Shape of oversampled X:")
print(X_resampled.shape)
print()
print("Shape of oversampled y:")
print(y_resampled.shape)
```

```
Shape of oversampled X:
(15224, 100)

Shape of oversampled y:
(15224,)
```

In [228…

```
df['StemmedText'] = stemmed_X
df.head()
```

Out[228…

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
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	

Machine Learning Algorithm (1st Part)

Logistic Regression

We have X-resampled, y_resampled -> Text input and Review (1/0 for positive/negative) for training input and output

In [229…
```
# Training set and test set:
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(12179, 100)
(3045, 100)
(12179,)
(3045,)
```

In [230…
```
lr = LogisticRegression(C=1e5)
result = lr.fit(X=X_train, y=y_train)
predictions = result.predict(X_test)
```

In [231…
```
predictions[:5]
```

Out[231…
```
array([0, 0, 0, 1, 1])
```

In [232…
```
from sklearn.metrics import precision_score, recall_score, f1_score
accuracy1 = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy1)
precision1 = precision_score(y_test, predictions)
print("Precision Score:", precision1)
recall1 = recall_score(y_test, predictions)
print("Recall Score:", recall1)
f1_score1 = f1_score(y_test, predictions)
print("F1 Score:", f1_score1)
```

```
Accuracy: 0.7885057471264367
Precision Score: 0.8250539956803455
Recall Score: 0.7407886231415644
F1 Score: 0.7806539509536785
```

In [233…
```
training_predictions = result.predict(X_train)
training_accuracy1 = accuracy_score(y_train, training_predictions)
print(training_accuracy1)
```

```
0.7998193611955005
```

In [234…
```
print(metrics.classification_report(y_test, predictions, target_names = ["pos
```

```
              precision    recall  f1-score   support

    positive       0.76      0.84      0.80      1498
    negative       0.83      0.74      0.78      1547

    accuracy                           0.79      3045
   macro avg       0.79      0.79      0.79      3045
weighted avg       0.79      0.79      0.79      3045
```

Confusion Matrix

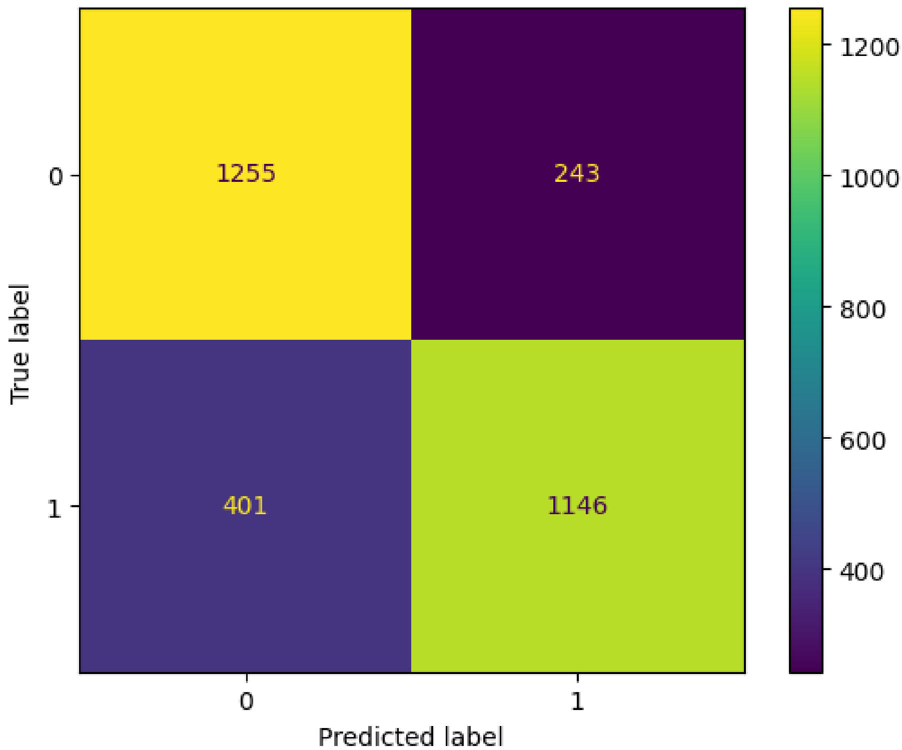
In [235…
```
cm1 = confusion_matrix(y_test, predictions, labels=lr.classes_)
```

```
print(cm1)
```

```
[[1255  243]
 [ 401 1146]]
```

In [236…

```
disp1 = ConfusionMatrixDisplay(confusion_matrix=cm1, display_labels=lr.classe
disp1.plot()
plt.show()
```



Adding Features and then Applying Logistic Regression

In [237…

```
df.head()
```

Out[237…

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres	1	

2	3	B000LQOCH0	ABXLMWJIXXAIN	"Natalia Corres"	1
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0

In [238…

```
final_y.head() # Our output column
```

Out[238…

```
0    1
1    0
2    1
3    0
4    1
Name: Review, dtype: int64
```

In [239…

```
multifeature_X = df[['WordCount',      'CharacterCount',      'HelpfulnessP
multifeature_X.head()
```

Out[239…

	WordCount	CharacterCount	HelpfulnessPercentage	HelpfulnessIndicator	Stemme
0	48	263	100.0	Useful	bought vital ca food pr fo
1	31	190	0.0	Not Available	produc label j salt p pear
2	94	509	100.0	Useful	c a centu pillowi
3	41	219	100.0	Useful	look i robi believ
4	27	140	0.0	Not Available	grea great wide yumm

In [240…

```
# Train test split
X_train, X_test, y_train, y_test = train_test_split(multifeature_X, final_y,
```

In [241…

```
X_train[:5]
```

Out[241…

	WordCount	CharacterCount	HelpfulnessPercentage	HelpfulnessIndicator	Stem
9529	17	78	100.000000	Useful	love g us
2169	75	373	0.000000	Not Available	hear ca
6270	66	361	0.000000	Not Available	wi I stuf
4781	107	581	82.352941	Useful	s use a (
8359	24	118	0.000000	Not Available	r r

In [242…

```
# Convert output y to one hot encoding if it's categorical - in our case, we

# Converting Helpfulness Indicator
encoder = OneHotEncoder()
X_train_encoded = encoder.fit_transform(X_train['HelpfulnessIndicator'].to_nu
X_test_encoded = encoder.transform(X_test['HelpfulnessIndicator'].to_numpy().
```

In [243…

```
print(X_train_encoded[:5])
print(X_train_encoded.shape)
```

```
  (0, 3)	1.0
  (1, 1)	1.0
  (2, 1)	1.0
  (3, 3)	1.0
  (4, 1)	1.0
(7990, 4)
```

In [244…

```
# Scaling the numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train[['WordCount',     'CharacterCou
X_test_scaled = scaler.transform(X_test[['WordCount',   'CharacterCount', 'He
```

In [245…

```
print(X_train_scaled[:5])
print(X_train_scaled.shape)
```

```
[[-0.82982417 -0.84079984  1.31511415]
 [-0.01064847 -0.0999254  -0.85245898]
 [-0.13776194 -0.13006267 -0.85245898]
```

```
 [ 0.44131053  0.42245386  0.93260124]
 [-0.73095813 -0.74034229 -0.85245898]]
(7990, 3)
```

In [246…

```
vectorizer = CountVectorizer(max_features=100)
X_train_text = vectorizer.fit_transform(X_train['StemmedText'])
X_test_text = vectorizer.transform(X_test['StemmedText'])
```

In [247…

```
print(X_train_text[:1])
print(X_train_text.shape)
```

```
  (0, 52)	1
  (0, 84)	1
  (0, 38)	1
  (0, 67)	1
  (0, 91)	1
(7990, 100)
```

In [248…

```
X_train_combined = hstack((X_train_encoded, X_train_scaled, X_train_text))
X_test_combined = hstack((X_test_encoded, X_test_scaled, X_test_text))
```

In [249…

```
print(X_train_combined.shape)
print(X_test_combined.shape)
```

```
(7990, 107)
(1998, 107)
```

Training Logistic Regression Model

In [250…

```
combined_result = lr.fit(X=X_train_combined, y=y_train)
predictions_with_FE = combined_result.predict(X_test_combined)
```

In [251…

```
accuracy2 = accuracy_score(y_test, predictions_with_FE)
print("Accuracy:", accuracy2)
precision2 = precision_score(y_test, predictions_with_FE)
print("Precision Score:", precision2)
recall2 = recall_score(y_test, predictions_with_FE)
print("Recall Score:", recall2)
f1_score2 = f1_score(y_test, predictions_with_FE)
print("F1 Score:", f1_score2)
```

```
Accuracy: 0.8073073073073073
Precision Score: 0.8161512027491409
Recall Score: 0.9570181329751511
F1 Score: 0.8809891808346213
```

In [252…

```
training_predictions_with_FE = combined_result.predict(X_train_combined)
training_accuracy2 = accuracy_score(y_train, training_predictions_with_FE)
print(training_accuracy2)
```

```
0.8086357947434293
```

In [253…

```
print(metrics.classification_report(y_test, predictions_with_FE, target_names
```

```
              precision    recall  f1-score   support

    positive       0.75      0.37      0.49       509
    negative       0.82      0.96      0.88      1489
```

```
    accuracy                           0.81      1998
   macro avg       0.78      0.66      0.69      1998
weighted avg       0.80      0.81      0.78      1998
```

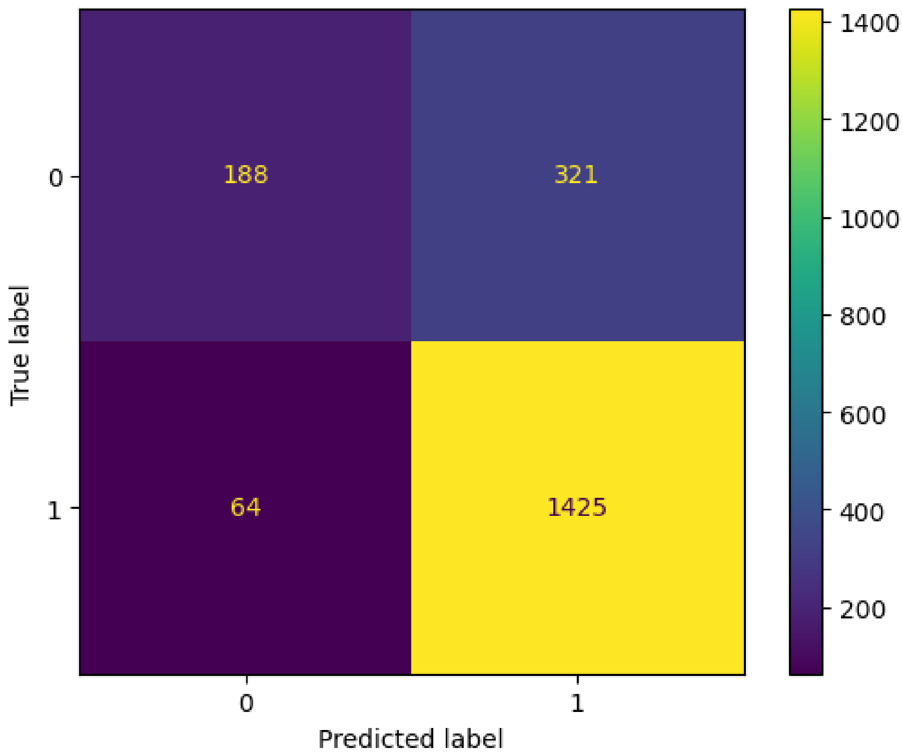
In [254…

```
cm2 = confusion_matrix(y_test, predictions_with_FE, labels=lr.classes_)
print(cm2)
```

```
[[ 188  321]
 [  64 1425]]
```

In [255…

```
disp2 = ConfusionMatrixDisplay(confusion_matrix=cm2, display_labels=lr.classe
disp2.plot()
plt.show()
```



PrettyTable

In [256…

```
# Table:
comparison_table = PrettyTable(["Model", "Test Accuracy", "Train Accuracy", "
comparison_table.add_row(["Logistic Regression with Text feature", round(accu
comparison_table.add_row(["Logistic Regression with Feature Engineering", rou
print(comparison_table)
```

```
+----------------------------------------------+---------------+--------------
-+-----------+--------+----------+
|                    Model                     | Test Accuracy | Train Accuracy
| Precision | Recall | F1 Score |
+----------------------------------------------+---------------+--------------
-+-----------+--------+----------+
|    Logistic Regression with Text feature     |     78.85     |     79.98
|   82.51   | 74.08  |  78.07   |
| Logistic Regression with Feature Engineering |     80.73     |     80.86
|   81.62   |  95.7  |   88.1   |
+----------------------------------------------+---------------+--------------
-+-----------+--------+----------+
```

Randomized Search Cross Validation

In [257…

```
# Concatenate the test and train variables back to perform randomizedsearchcv
print(y_train.shape)
print(y_test.shape)
X_with_FE = vstack((X_train_combined, X_test_combined))
y_with_FE = np.concatenate((y_train, y_test))
print(y_with_FE.shape)
```

```
(7990,)
(1998,)
(9988,)
```

In [259…

```
# model = LogisticRegression()
# cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# space = dict()
# space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
# space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
# space['C'] = loguniform(1e-5, 100)

# search = RandomizedSearchCV(model, space, n_iter=500, scoring='accuracy', n


# rscv_result = search.fit(X_with_FE, y_with_FE)

# print('Best Score: %s' % rscv_result.best_score_)
# print('Best Hyperparameters: %s' % rscv_result.best_params_)
```

In [260…

```
print('Best Hyperparameters: %s' % rscv_result.best_params_)
```

```
Best Hyperparameters: {'C': 0.18259106330120106, 'penalty': 'l2', 'solver': 'ne
wton-cg'}
```

In [261…

```
rscv_model = LogisticRegression(C=0.182591063301201, penalty='l2', solver='ne

rscv_model_result = rscv_model.fit(X=X_train_combined, y=y_train)
rscv_predictions_with_FE = combined_result.predict(X_test_combined)

accuracy_rscv = round(accuracy_score(y_test, rscv_predictions_with_FE)*100,2)
precision_rscv = round(precision_score(y_test, rscv_predictions_with_FE)*100,
recall_rscv = round(recall_score(y_test, rscv_predictions_with_FE)*100,2)
f1_score_rscv = round(f1_score(y_test, rscv_predictions_with_FE)*100,2)

# Training accuracy:
rscv_train_predictions = combined_result.predict(X_train_combined)
train_accuracy_rscv = round(accuracy_score(y_train, rscv_train_predictions)*1

comparison_table.add_row(["Feature Engineering with RandomizedSearchCV", accu
print(comparison_table)
```

```
+--------------------------------------------+---------------+---------------
-+-----------+--------+----------+
|                   Model                    | Test Accuracy | Train Accuracy
| Precision | Recall | F1 Score |
+--------------------------------------------+---------------+---------------
-+-----------+--------+----------+
|    Logistic Regression with Text feature    |     78.85     |     79.98
|   82.51   | 74.08  |  78.07   |
| Logistic Regression with Feature Engineering |     80.73     |     80.86
|   81.62   |  95.7  |   88.1   |
| Feature Engineering with RandomizedSearchCV  |     80.73     |     80.86
```

```
|   81.62    |  95.7  |   88.1   |
+--------------------------------------------+---------------+--------------
-+-----------+--------+----------+
```

Grid Search Cross Validation

The main difference is that the search space must be a discrete grid to be searched. This means that instead of using a log-uniform distribution for C, we can specify discrete values on a log scale.

In [263…

```
# model = LogisticRegression()

# cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# space = dict()
# space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
# space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
# space['C'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]

# search = GridSearchCV(model, space, scoring='accuracy', n_jobs=-1, cv=cv)

# gscv_result = search.fit(X_with_FE, y_with_FE)

# print('Best Score: %s' % gscv_result.best_score_)
# print('Best Hyperparameters: %s' % gscv_result.best_params_)
```

In [264…

```
print('Best Hyperparameters: %s' % gscv_result.best_params_)
```

```
Best Hyperparameters: {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
```

In [265…

```
gscv_model = LogisticRegression(C=1, penalty='l1', solver='liblinear')

gscv_model_result = gscv_model.fit(X=X_train_combined, y=y_train)
gscv_predictions_with_FE = combined_result.predict(X_test_combined)

accuracy_gscv = round(accuracy_score(y_test, gscv_predictions_with_FE)*100,2)
precision_gscv = round(precision_score(y_test, gscv_predictions_with_FE)*100,
recall_gscv = round(recall_score(y_test, gscv_predictions_with_FE)*100,2)
f1_score_gscv = round(f1_score(y_test, gscv_predictions_with_FE)*100,2)

# Training accuracy:
gscv_train_predictions = combined_result.predict(X_train_combined)
train_accuracy_gscv = round(accuracy_score(y_train, gscv_train_predictions)*1

comparison_table.add_row(["Feature Engineering with GridSearchCV", accuracy_g
print(comparison_table)
```

```
+--------------------------------------------+---------------+--------------
-+-----------+--------+----------+
|                   Model                    | Test Accuracy | Train Accuracy
| Precision | Recall | F1 Score |
+--------------------------------------------+---------------+--------------
-+-----------+--------+----------+
|    Logistic Regression with Text feature    |     78.85     |     79.98
|   82.51    | 74.08  |  78.07   |
| Logistic Regression with Feature Engineering |     80.73     |     80.86
|   81.62    |  95.7  |   88.1   |
| Feature Engineering with RandomizedSearchCV |     80.73     |     80.86
|   81.62    |  95.7  |   88.1   |
|    Feature Engineering with GridSearchCV    |     80.73     |     80.86
|   81.62    |  95.7  |   88.1   |
+--------------------------------------------+---------------+--------------
```

```
-+-----------+--------+----------+
```

In [266…

```
import time

start = time.time()

time.sleep(5)
time.sleep(2)


end = time.time()-start
print(end)
```

```
7.007612705230713
```

In [267…

```
# Training set and test set:
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
```

In [268…

```
X_resampled
```

Out[268…

```
<15224x100 sparse matrix of type '<class 'numpy.int64'>'
	with 170811 stored elements in Compressed Sparse Row format>
```

In [269…

```
y_resampled
```

Out[269…

```
0        1
1        0
2        1
3        0
4        1
        ..
15219    0
15220    0
15221    0
15222    0
15223    0
Name: Review, Length: 15224, dtype: int64
```

(1) KNN

In [270…

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

In [271…

```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
```

In [272…

```
knn_classifier = KNeighborsClassifier(n_neighbors=3)
```

In [273…

```
knn_classifier.fit(X_train, y_train)
```

Out[273…

```
KNeighborsClassifier(n_neighbors=3)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

page with nbviewer.org.

In [274…
```
y_pred = knn_classifier.predict(X_test)
```

In [275…
```
y_pred
```

Out[275…
```
array([0, 0, 0, ..., 0, 0, 1])
```

In [276…
```
accuracy = accuracy_score(y_test, y_pred)
```

In [277…
```
accuracy
```

Out[277…
```
0.670935960591133
```

(2) Naive Bayes classifier

In [278…
```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB

from sklearn import metrics
```

In [279…
```
# GaussianNB
```

In [280…
```
G_classifier = GaussianNB()
```

In [281…
```
X_train = X_train.toarray()
```

In [282…
```
G_classifier.fit(X_train, y_train)
```

Out[282…
```
GaussianNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [283…
```
X_test = X_test.toarray()

predictions_G = G_classifier.predict(X_test)
```

In [284…
```
predictions_G
```

Out[284…
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [285…
```
accuracy = metrics.accuracy_score(y_test, predictions_G)
```

In [286…
```
accuracy
```

```
accuracy
```

Out[286… 0.6949096880131362

In [287…
```
# BernoulliNB
```

In [288…
```
B_classifier = BernoulliNB()
```

In [289…
```
B_classifier.fit(X_train, y_train)
```

Out[289… BernoulliNB()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [290…
```
predictions_B = B_classifier.predict(X_test)
```

In [291…
```
predictions_B
```

Out[291… array([0, 0, 0, ..., 1, 0, 0])

In [292…
```
accuracy_B = metrics.accuracy_score(y_test, predictions_B)
```

In [293…
```
accuracy_B
```

Out[293… 0.7408866995073892

In [294…
```
# MultinomialNB
```

In [295…
```
M_classifier = MultinomialNB()
```

In [296…
```
M_classifier.fit(X_train, y_train)
```

Out[296… MultinomialNB()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [297…
```
predictions_M = M_classifier.predict(X_test)
```

In [298…
```
predictions_M
```

Out[298… array([1, 1, 0, ..., 0, 0, 1])

In [299…
```
accuracy_M = metrics.accuracy_score(y_test, predictions_M)
```

In [300…
```
accuracy_M
```
Out[300…
```
0.7487684729064039
```

(3) Decision Tree

In [301…
```
from sklearn.tree import DecisionTreeClassifier
```
In [302…
```
clf = DecisionTreeClassifier()
```
In [303…
```
clf.fit(X_train, y_train)
```
Out[303…
```
DecisionTreeClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [304…
```
y_pred = clf.predict(X_test)
```
In [305…
```
accuracy = metrics.accuracy_score(y_test, y_pred)
```
In [306…
```
accuracy
```
Out[306…
```
0.7770114942528735
```

(4) Random Forest

In [307…
```
from sklearn.ensemble import RandomForestClassifier
```
In [308…
```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```
In [309…
```
rf_classifier.fit(X_train, y_train)
```
Out[309…
```
RandomForestClassifier(random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [310…
```
y_pred = rf_classifier.predict(X_test)
```
In [311…
```
accuracy = metrics.accuracy_score(y_test, y_pred)
```

In [312…
```
accuracy
```

Out[312…
```
0.8390804597701149
```

(5) Boosting Algorithm

In [313…
```
from sklearn.ensemble import AdaBoostClassifier
```

In [314…
```
base_classifier = DecisionTreeClassifier(max_depth=1)
```

In [315…
```
adaboost_classifier = AdaBoostClassifier(base_classifier, n_estimators=50, ra
```

In [316…
```
adaboost_classifier.fit(X_train, y_train)
```

Out[316…
```
AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1),
                   random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [317…
```
y_pred = adaboost_classifier.predict(X_test)
```

In [318…
```
accuracy = metrics.accuracy_score(y_test, y_pred)
```

In [319…
```
accuracy
```

Out[319…
```
0.7835796387520525
```

(6). Logistic Regression

In [320…
```
from sklearn import linear_model
```

In [321…
```
lrg = linear_model.LogisticRegression()
```

In [322…
```
lrg.fit(X_train, y_train)
```

Out[322…
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [323…
```
train_predictions = lrg.predict(X_train)

train_accuracy7 = accuracy_score(y_train, train_predictions)
```

In [324…
```
test_predictions = lrg.predict(X_test)

test_accuracy7 = accuracy_score(y_test, test_predictions)
```

In [325…
```
print(f"Training Accuracy: {train_accuracy7}")
print(f"Testing Accuracy: {test_accuracy7}")
```

```
Training Accuracy: 0.7999835782905
Testing Accuracy: 0.7881773399014779
```

(7).Linear Regression

In [326…
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [327…
```
model = LinearRegression()
```

In [328…
```
model.fit(X_train, y_train)
```

Out[328…
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [329…
```
train_predictions = clf.predict(X_train)

train_accuracy8 = accuracy_score(y_train, train_predictions)
```

In [330…
```
test_predictions = clf.predict(X_test)

test_accuracy8 = accuracy_score(y_test, test_predictions)
```

In [331…
```
print(f"Training Accuracy: {train_accuracy8}")
print(f"Testing Accuracy: {test_accuracy8}")
```

```
Training Accuracy: 0.996387223910009
Testing Accuracy: 0.7770114942528735
```

(8).Gradient Boosting Machines (GBM)

In [332…
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [333…
```
model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_c
```

In [334…
```
model.fit(X_train, y_train)
```

Out[334…
```
GradientBoostingClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [335…
```
train_predictions = model.predict(X_train)

train_accuracy9 = accuracy_score(y_train, train_predictions)
```

In [336…
```
test_predictions = model.predict(X_test)

test_accuracy9 = accuracy_score(y_test, test_predictions)
```

In [337…
```
print(f"Training Accuracy: {train_accuracy9}")
print(f"Testing Accuracy: {test_accuracy9}")
```

```
Training Accuracy: 0.8135314886279662
Testing Accuracy: 0.7885057471264367
```