INDIAN INSTITUTE OF INFORMATION TECHNOLOGY DHARWAD

Dharwad, Karnataka- 580009

A project report on

"Analysis Of Product Review Using Sentiment Analysis"

Submitted By

Tukuntla Venkata Hemanth (19BCS107)
Mani Deep Venigalla (19BCS112)
R Mounika(19BCS124)
Ravi Shankar Sharma(19BCS091)

R. Chandra Vardhan(19BCS092)

Under the guidance of

Dr. Deepak K T

Asst.Professor, Department of ECE

DECLARATION

-

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Tukuntla Venkata Hemanth
19BCS107

Mani Deep Venigalla
19BCS112

R Mounika
19BCS124

Ravi Shankar Sharma
19BCS091

R. Chandra Vardhan
19BCS092

Acknowledgement:

We would like to express our sincere gratitude to Dr. Deepak K T, Asst.Professor, Department of CSE-IIIT Dharwad for her guidance and constant support throughout the course of this mini project. We would also like to thank all the faculty and administration of the institute who ensured the needs fulfilled for the completion of this project. We would also like to thank our families and friends for encouraging us and giving us moral and emotional support during the transition from online mode to offline mode.

Place: Dharwad

Tukuntla Venkata Hemanth (19BCS107)
Mani Deep Venigalla (19BCS112)
R Mounika (19BCS124)
Ravi Shanakar Sharma (19BCS091)

R. Chandra Vardhan (19BCS092)

Department of CSE and ECE, IIIT Dharwad

Abstract:

We have considered a data set from Kaggle Amazon Reviews Dataset. This data set contains data about the customer reviews of the amazon products. The data set is a .csv file containing reviewtext, review ratings, of the customers along with their review id, review province etc, and their coordinates.

We have done all the preprocessing required on the data set, added a column sentiment which decides the positive or negative review to the dataset based on review rating and also we used nltk packages to text cleaning and removing unwanted words for deciding the reviews(stopwords).

After the preprocessing phase we used bag of words strategy to convert the text content in reviews to numerical feature vectors, for that we used SciKit-Learn's **CountVectorizer** and also we use **TfidfTransformer** to overshadow the words which has lower average counts with same frequencies ,which has very little meaning

We split the train and test data and train using machine learning models after defining the feature vector values with CountVectorizer and TfidfTransformer. To determine the model performance, use Naive Bayes analysis, Support Vector Machines, and Logistic Regression, and print the confusion matrix and accuracy.

Contents:

Chapter 1

Introduction

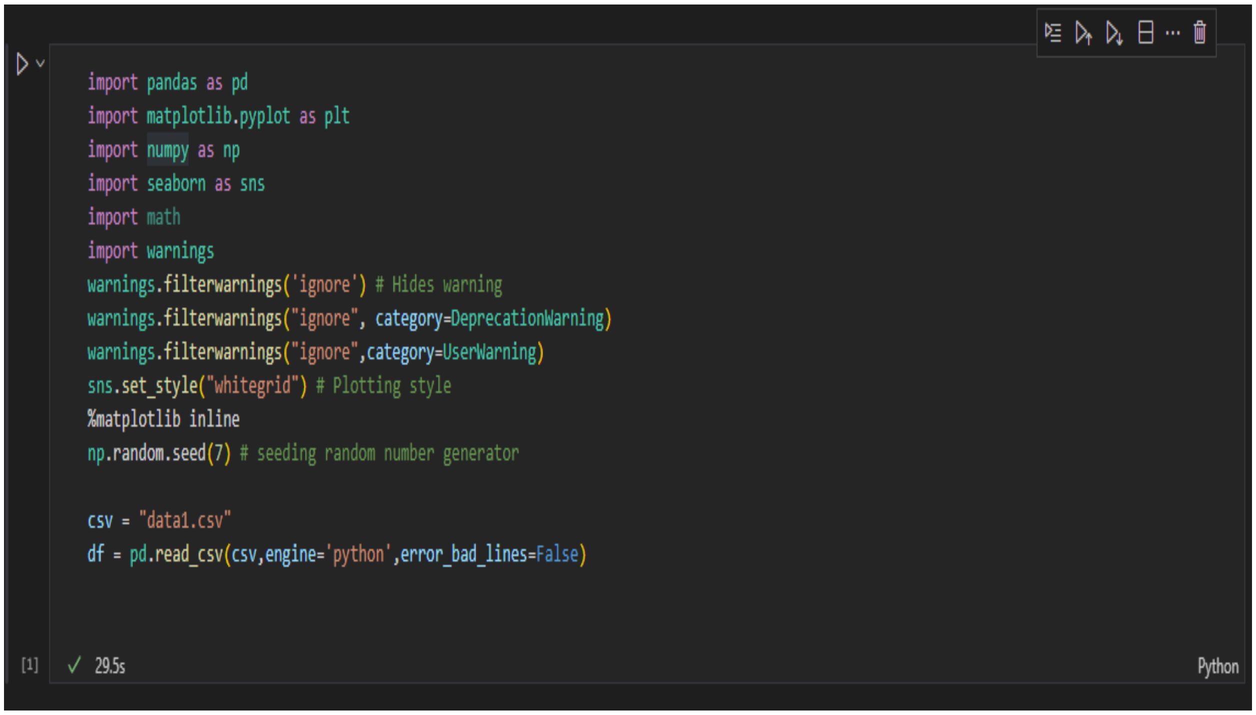
Sentiment analysis (also known as opinion mining) is a natural language processing **(NLP)** approach for determining the positive, negative, or neutral nature of data.

The application problem of sentiment analysis of product evaluations has lately gained a lot of traction in text mining and computational linguistics research. We're looking to see if there's a link between Amazon product reviews and consumer ratings. We employ Naive Bayes analysis, Support Vector Machines, and Logistic Regression as well as other classic machine learning techniques. We may gain a better understanding of these algorithms by comparing the outcomes. They might also be used in conjunction with other fraud detection approaches.

Chapter 2

Importing packages and loading data

We are importing pandas to read the .csv data files of product reviews and matplotlib and sns for plotting the graphs and bar plots and warnings to ignore warnings

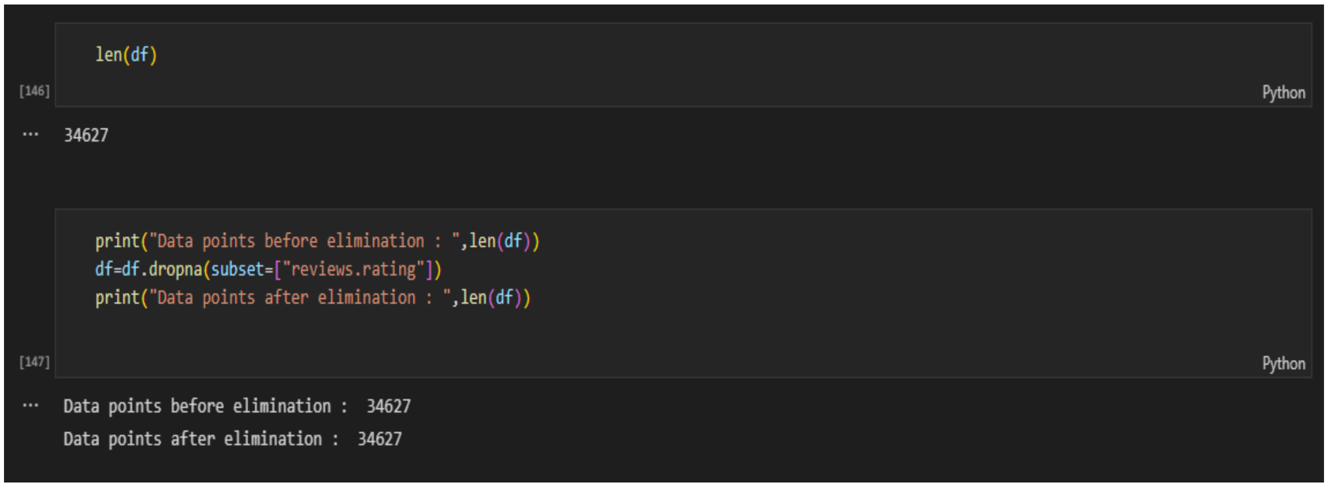


```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import math
import warnings
warnings.filterwarnings('ignore') # Hides warning
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore",category=UserWarning)
sns.set_style("whitegrid") # Plotting style
%matplotlib inline
np.random.seed(7) # seeding random number generator

csv = "data1.csv"
df = pd.read_csv(csv,engine='python',error_bad_lines=False)
```

Chapter 3

<u>Dataset-Description</u>

Our dataset comes from **Consumer Reviews of Amazon Products** which is taken from kaggle website. This dataset has **34627** data points in total. Each example includes the type of product, name of the product as well as the text review and the rating of the product. To better utilize the data, first we extract the rating and review column since these two are the essential part of this project. Then we tried to delete some data points which has no ratings through the data we found that the dataset is good as it has no NULL values in the review ratings so we have **34627** data points in **total**.



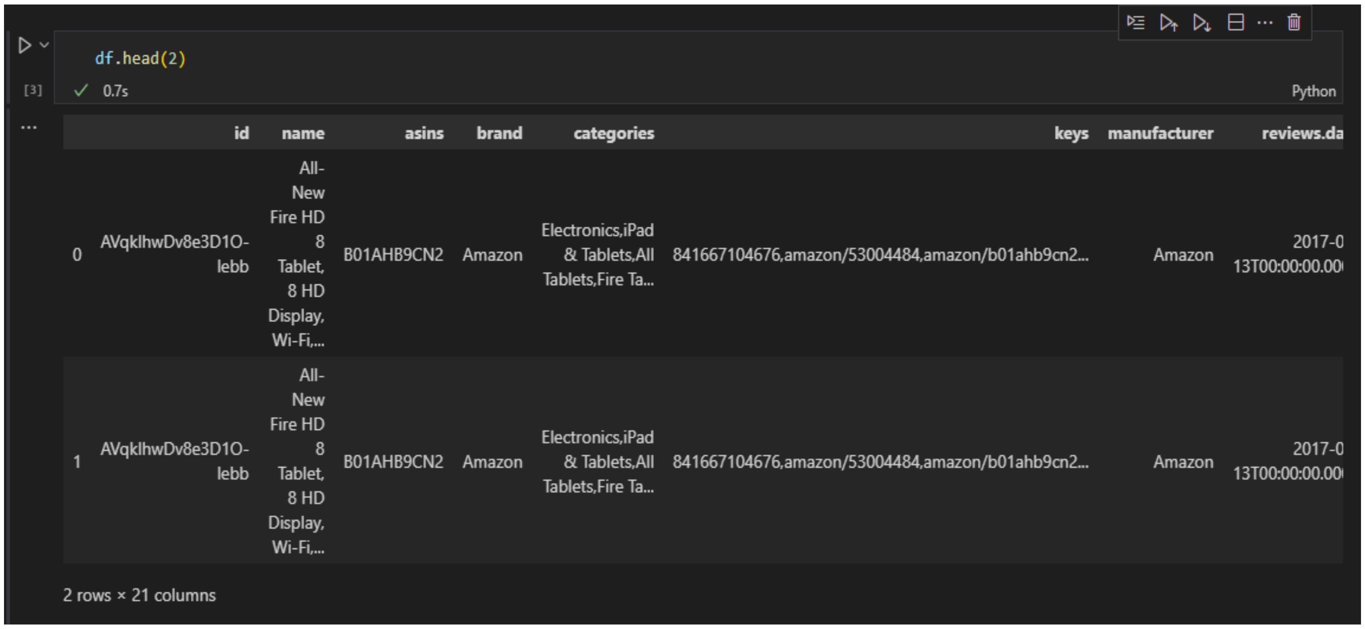
```
len(df)
```
[146] Python

```
34627
```

```
print("Data points before elimination : ",len(df))
df=df.dropna(subset=["reviews.rating"])
print("Data points after elimination : ",len(df))
```
[147] Python

```
Data points before elimination :  34627
Data points after elimination :  34627
```

The Dataset have 21 columns on a total that are id, asins(**Amazon Standard Identification Number**),brand,categories,keys, manufacturer, reviews.date, reviews.dateAdded ,reviews.dateSeen ,reviews.didPurchase, reviews.doRecommend, reviews.id ,reviews.numHelpful ,reviews.rating, reviews.sourceURLs, reviews.text, reviews.title ,reviews.userCity, reviews.userProvince ,manufacturerNumber ,sourceURLs .



```
df.head(2)
```
[3] ✓ 0.7s Python

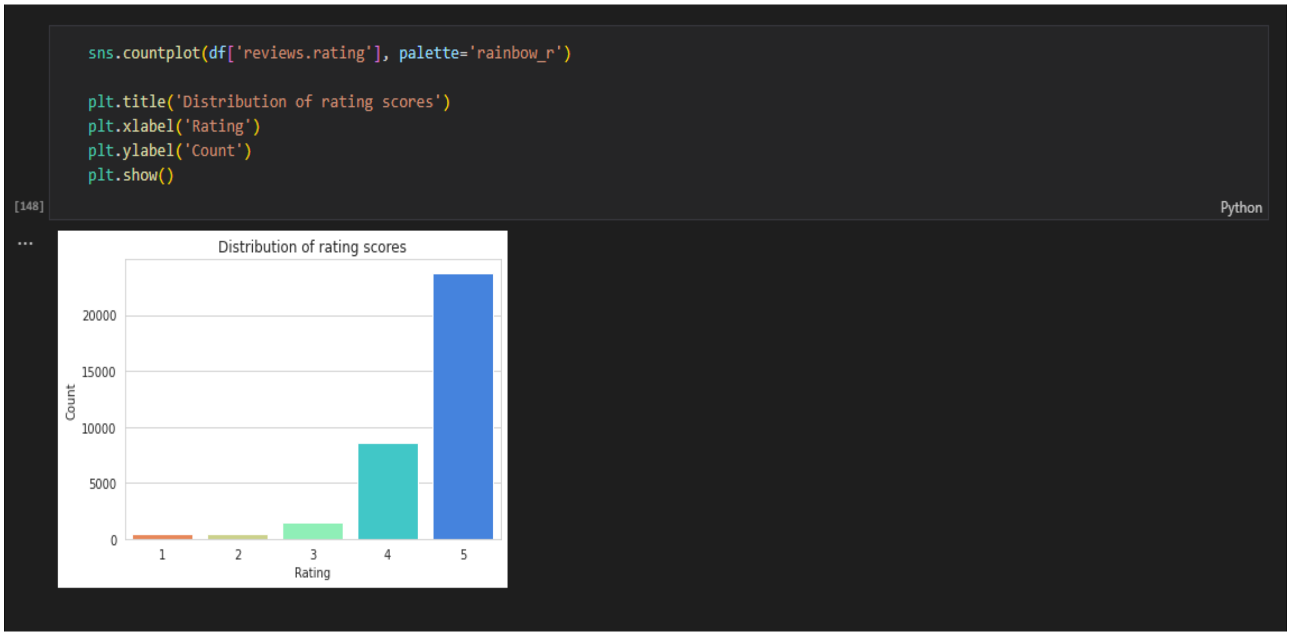
	id	name	asins	brand	categories	keys	manufacturer	reviews.da
0	AVqklhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi,...	B01AHB9CN2	Amazon	Electronics,iPad & Tablets,All Tablets,Fire Ta...	841667104676,amazon/53004484,amazon/b01ahb9cn2...	Amazon	2017-0 13T00:00:00.00
1	AVqklhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi,...	B01AHB9CN2	Amazon	Electronics,iPad & Tablets,All Tablets,Fire Ta...	841667104676,amazon/53004484,amazon/b01ahb9cn2...	Amazon	2017-0 13T00:00:00.00

2 rows × 21 columns

Chapter 4

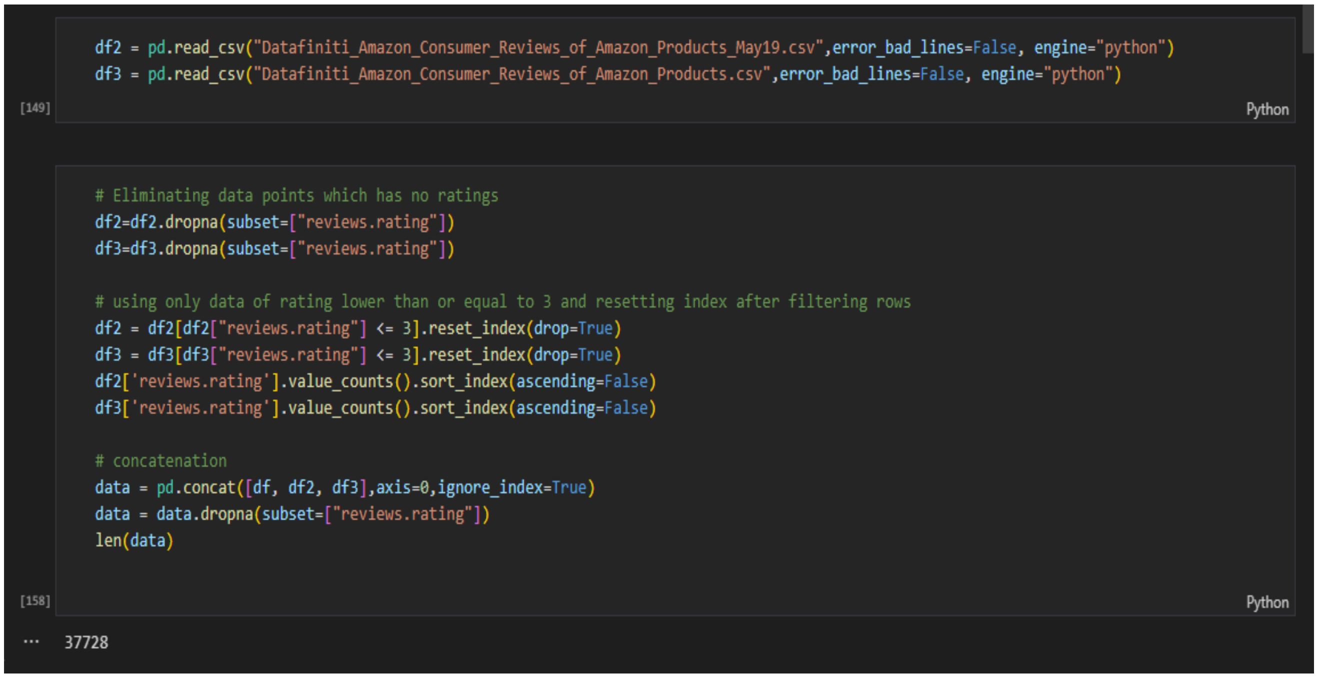
Exploratory Data Analysis (EDA)

We showed the distribution of the ratings to get a quick overview of the dataset. It demonstrates that we have five classes with ratings ranging from 1 to 5, as well as the distribution of those classes. Furthermore, these five classes are really unbalanced, with class 1 and 2 having minimal quantities of data compared to class 5, which has over 20000 reviews.



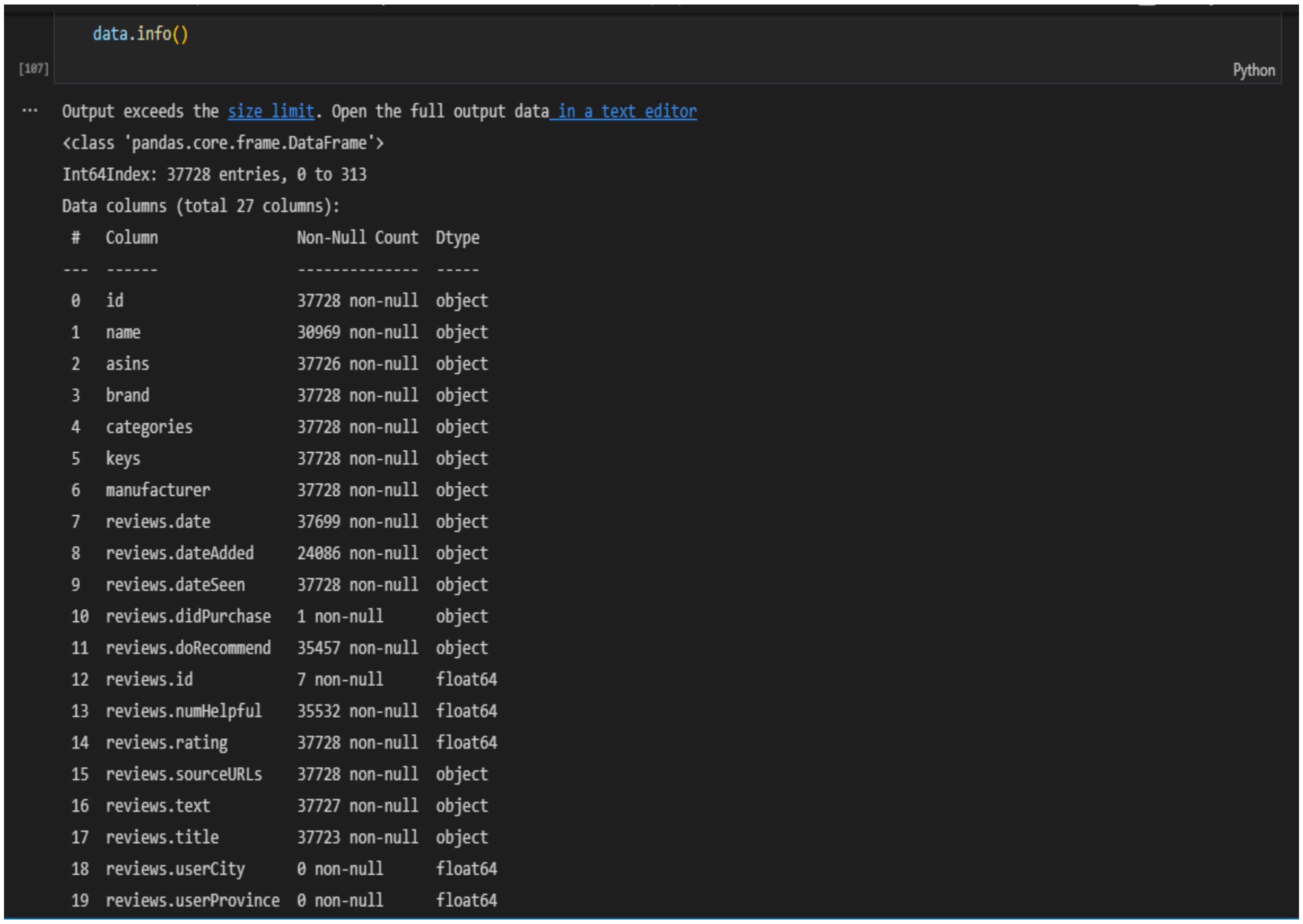
```
sns.countplot(df['reviews.rating'], palette='rainbow_r')

plt.title('Distribution of rating scores')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()
```

Due to the high imbalance of our dataset, we found and added more data points with low ratings from other resources. After adding the data points the total length of the data is **37728**



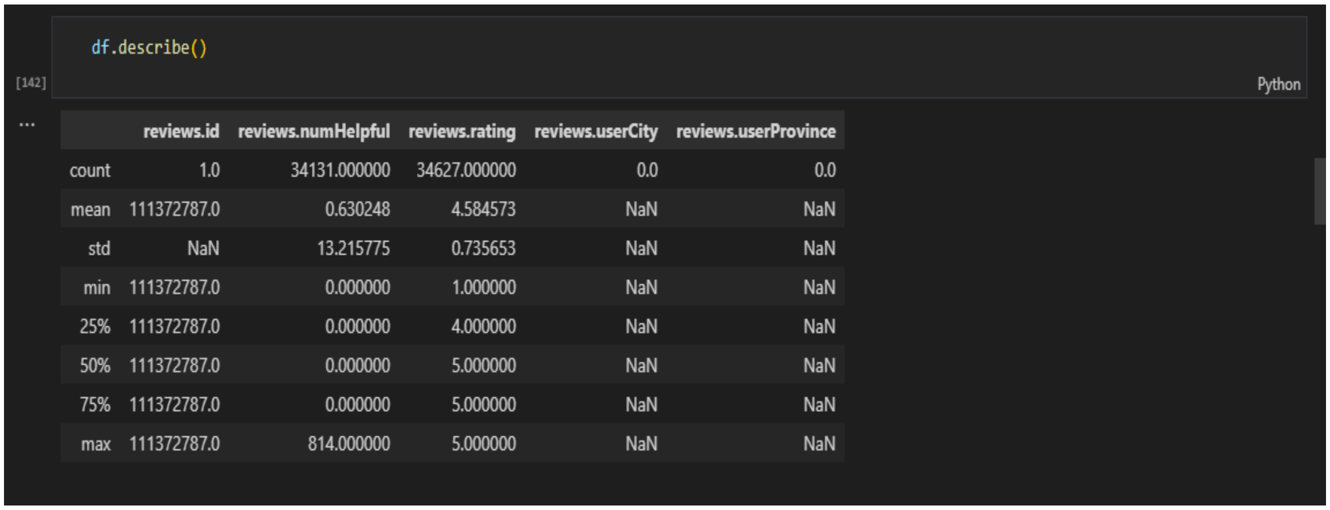
```
df2 = pd.read_csv("Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products_May19.csv",error_bad_lines=False, engine="python")
df3 = pd.read_csv("Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products.csv",error_bad_lines=False, engine="python")
[149]                                                                                                          Python

# Eliminating data points which has no ratings
df2=df2.dropna(subset=["reviews.rating"])
df3=df3.dropna(subset=["reviews.rating"])

# using only data of rating lower than or equal to 3 and resetting index after filtering rows
df2 = df2[df2["reviews.rating"] <= 3].reset_index(drop=True)
df3 = df3[df3["reviews.rating"] <= 3].reset_index(drop=True)
df2['reviews.rating'].value_counts().sort_index(ascending=False)
df3['reviews.rating'].value_counts().sort_index(ascending=False)

# concatenation
data = pd.concat([df, df2, df3],axis=0,ignore_index=True)
data = data.dropna(subset=["reviews.rating"])
len(data)
[158]                                                                                                          Python
...  37728
```

We used this strategy to tackle the problem of data imbalance which was present previously.



```
data.info()
[107]                                                                                                          Python

...  Output exceeds the size limit. Open the full output data in a text editor
     <class 'pandas.core.frame.DataFrame'>
     Int64Index: 37728 entries, 0 to 313
     Data columns (total 27 columns):
      #   Column                Non-Null Count  Dtype
     ---  ------                --------------  -----
      0   id                    37728 non-null  object
      1   name                  30969 non-null  object
      2   asins                 37726 non-null  object
      3   brand                 37728 non-null  object
      4   categories            37728 non-null  object
      5   keys                  37728 non-null  object
      6   manufacturer          37728 non-null  object
      7   reviews.date          37699 non-null  object
      8   reviews.dateAdded     24086 non-null  object
      9   reviews.dateSeen      37728 non-null  object
      10  reviews.didPurchase   1 non-null      object
      11  reviews.doRecommend   35457 non-null  object
      12  reviews.id            7 non-null      float64
      13  reviews.numHelpful    35532 non-null  float64
      14  reviews.rating        37728 non-null  float64
      15  reviews.sourceURLs    37728 non-null  object
      16  reviews.text          37727 non-null  object
      17  reviews.title         37723 non-null  object
      18  reviews.userCity      0 non-null      float64
      19  reviews.userProvince  0 non-null      float64
```

We dropped the columns reviews.userCity, reviews.userProvince, reviews.id, and reviews.didPurchase since these values are floats and are used for exploratory data analysis only and also not every category (column) have maximum number of values in comparison to total number of values(37728).We focus on the reviews.text as we need to decide the sentiment and also reviews.text category has minimum missing data (37727/37728) which is good for proceeding forward



```
df.describe()
```

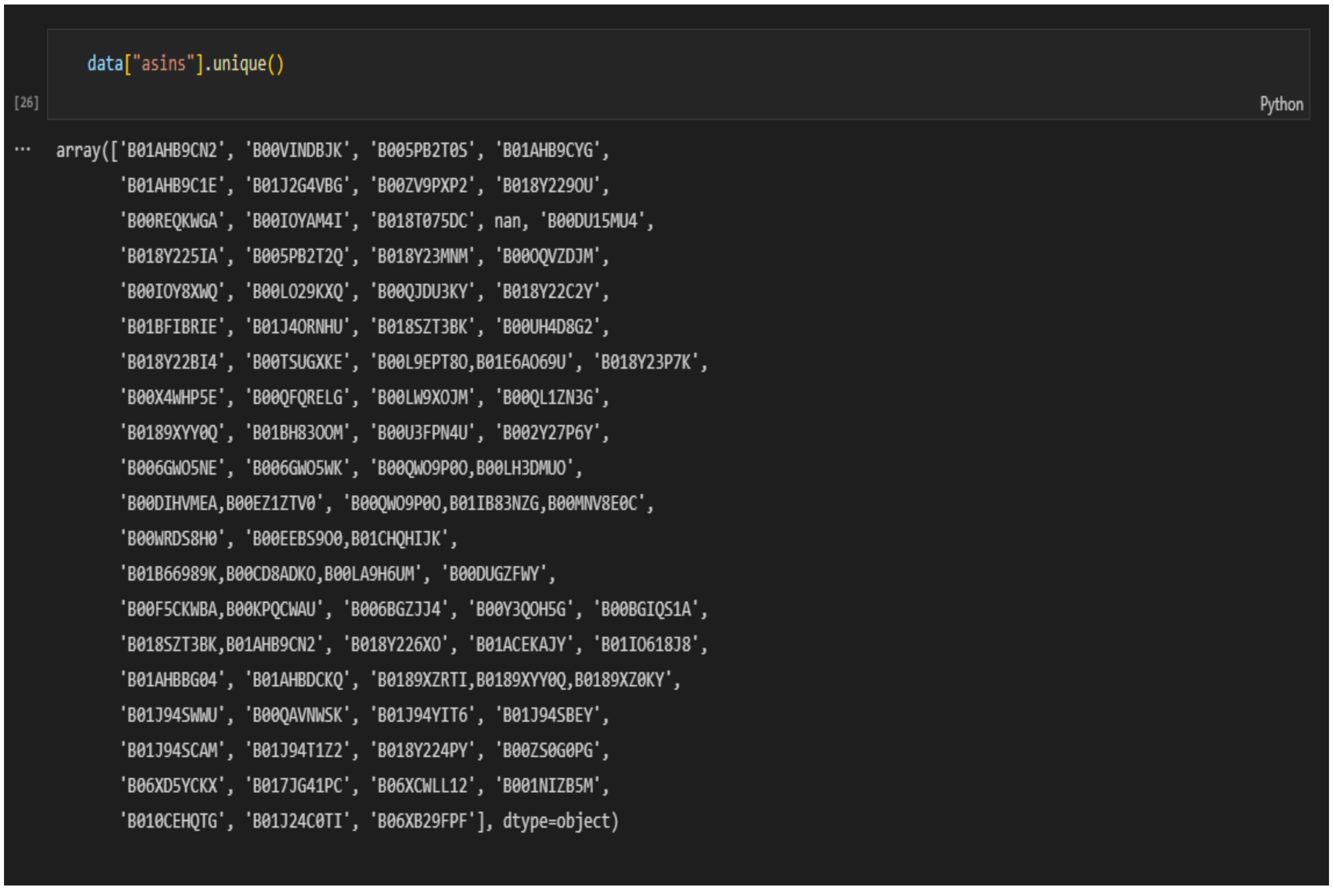
[142] Python

	reviews.id	reviews.numHelpful	reviews.rating	reviews.userCity	reviews.userProvince
count	1.0	34131.000000	34627.000000	0.0	0.0
mean	111372787.0	0.630248	4.584573	NaN	NaN
std	NaN	13.215775	0.735653	NaN	NaN
min	111372787.0	0.000000	1.000000	NaN	NaN
25%	111372787.0	0.000000	4.000000	NaN	NaN
50%	111372787.0	0.000000	5.000000	NaN	NaN
75%	111372787.0	0.000000	5.000000	NaN	NaN
max	111372787.0	814.000000	5.000000	NaN	NaN

Based on the descriptive statistics above, we see the following:

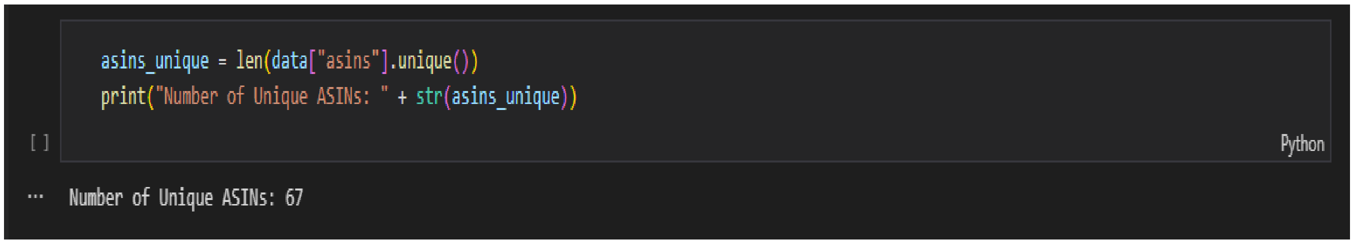
Average review score is 4.38, with low standard deviation which means most of the reviews are positive from 2nd quartile onwards (most part of the area is in 4 and 5)also the average for number of reviews helpful (reviews.numHelpful) is 0.65 but high standard deviation. The data are pretty spread out around the mean, and since you can't have negative people finding something helpful, then this is only on the right tail side. The range of most reviews will be between 0-13 people finding helpful (reviews.numHelpful)

812 people found the most helpful review to be useful. This might be a thorough, in-depth assessment that's well worth analyzing upon.



```
[26]
data["asins"].unique()
```

```
... array(['B01AHB9CN2', 'B00VINDBJK', 'B005PB2T0S', 'B01AHB9CYG',
          'B01AHB9C1E', 'B01J2G4VBG', 'B00ZV9PXP2', 'B018Y229OU',
          'B00REQKWGA', 'B00IOYAM4I', 'B018T075DC', nan, 'B00DU15MU4',
          'B018Y225IA', 'B005PB2T2Q', 'B018Y23MNM', 'B00OQVZDJM',
          'B00IOY8XWQ', 'B00LO29KXQ', 'B00QJDU3KY', 'B018Y22C2Y',
          'B01BFIBRIE', 'B01J4ORNHU', 'B018SZT3BK', 'B00UH4D8G2',
          'B018Y22BI4', 'B00TSUGXKE', 'B00L9EPT8O,B01E6AO69U', 'B018Y23P7K',
          'B00X4WHP5E', 'B00QFQRELG', 'B00LW9XOJM', 'B00QL1ZN3G',
          'B0189XYY0Q', 'B01BH83OOM', 'B00U3FPN4U', 'B002Y27P6Y',
          'B006GWO5NE', 'B006GWO5WK', 'B00QWO9P0O,B00LH3DMUO',
          'B00DIHVMEA,B00EZ1ZTV0', 'B00QWO9P0O,B01IB83NZG,B00MNV8E0C',
          'B00WRDS8H0', 'B00EEBS9O0,B01CHQHIJK',
          'B01B66989K,B00CD8ADKO,B00LA9H6UM', 'B00DUGZFWY',
          'B00F5CKWBA,B00KPQCWAU', 'B006BGZJJ4', 'B00Y3QOH5G', 'B00BGIQS1A',
          'B018SZT3BK,B01AHB9CN2', 'B018Y226XO', 'B01ACEKAJY', 'B01IO618J8',
          'B01AHBBG04', 'B01AHBDCKQ', 'B0189XZRTI,B0189XYY0Q,B0189XZ0KY',
          'B01J94SWWU', 'B00QAVNWSK', 'B01J94YIT6', 'B01J94SBEY',
          'B01J94SCAM', 'B01J94T1Z2', 'B018Y224PY', 'B00ZS0G0PG',
          'B06XD5YCKX', 'B017JG41PC', 'B06XCWLL12', 'B001NIZB5M',
          'B010CEHQTG', 'B01J24C0TI', 'B06XB29FPF'], dtype=object)
```



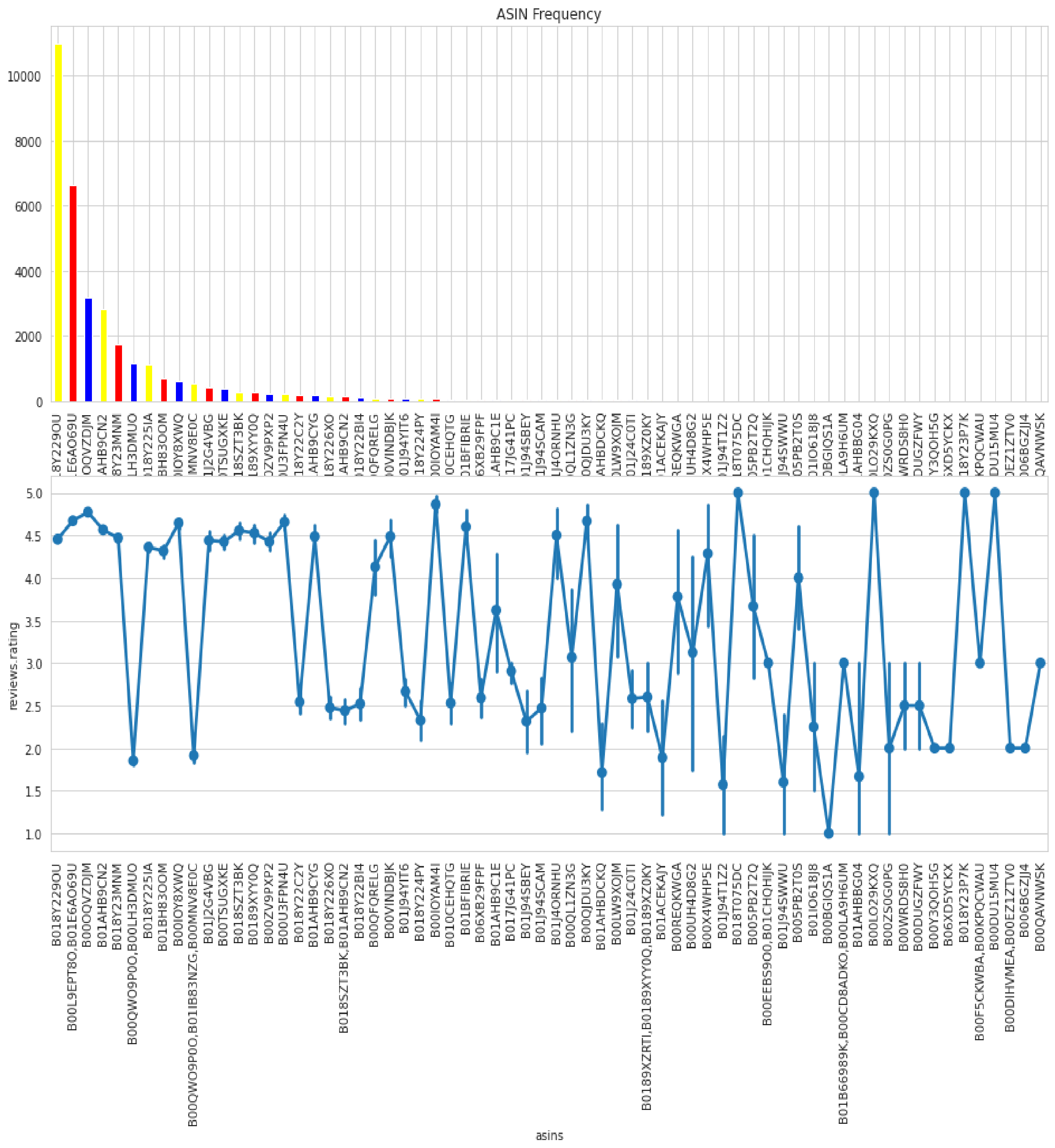
```
asins_unique = len(data["asins"].unique())
print("Number of Unique ASINs: " + str(asins_unique))
```

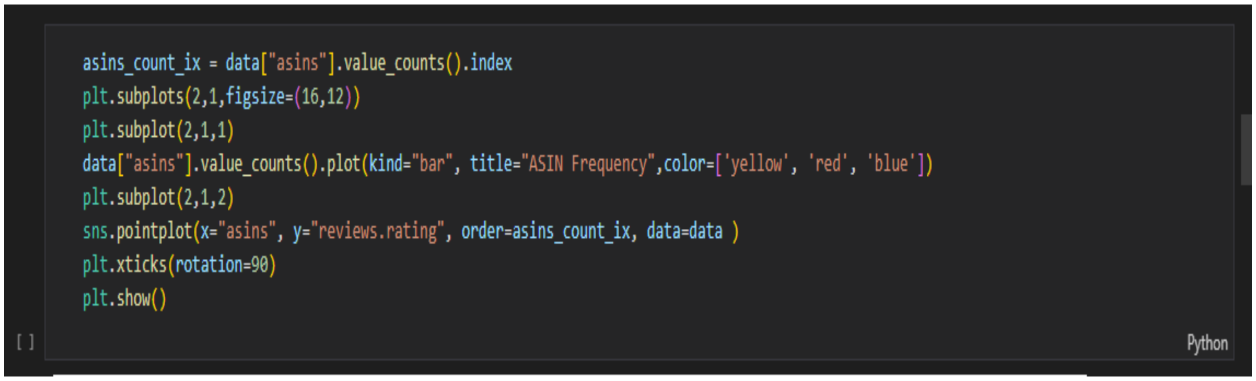
```
... Number of Unique ASINs: 67
```

Number of Unique ASINs in Dataset : 72
Following that, we'll look at the following columns:

- **asins**
- **reviews.rating**

- (**reviews.numHelpful** - not possible since numHelpful is only between 0-13 as per previous analysis in Raw Data)
- **reviews.text** - not possible since text is in long words)


ASIN Frequency
0
2000
4000
6000
8000
10000
reviews.rating
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
asins
B018Y229OU
B00L9EPT8O,B01E6AO69U
B000QVZDJM
B01AHB9CN2
B018Y23MNM
B00QWO9P0O,B00LH3DMUO
B018Y225IA
B01BH83OOM
B00IOY8XWQ
B00QWO9P0O,B01IB83NZG,B00MNV8E0C
B01J2G4VBG
B00TSUGXKE
B018SZT3BK
B0189XYY0Q
B00ZV9PXP2
B00U3FPN4U
B018Y22C2Y
B01AHB9CYG
B018Y226XO
B018SZT3BK,B01AHB9CN2
B018Y22BI4
B00QFQRELG
B00VINDBJK
B01J94YIT6
B018Y224PY
B00IOYAM4I
B010CEHQTG
B01BFIBRIE
B06XB29FPF
B01AHB9C1E
B017JG41PC
B01J94SBEY
B01J94SCAM
B01J4ORNHU
B00QL1ZN3G
B00QJDU3KY
B01AHBDCKQ
B00LW9XOJM
B01J24C0TI
B0189XZRTI,B0189XYY0Q,B0189XZ0KY
B01ACEKAJY
B00REQKWGA
B00UH4D8G2
B00X4WHP5E
B01J94T1Z2
B018T075DC
B005PB2T2Q
B00EEBS9O0,B01CHQHIJK
B01J94SWWU
B005PB2T0S
B01IO618J8
B00BGIQS1A
B01B66989K,B00CD8ADKO,B00LA9H6UM
B01AHBBG04
B00LO29KXQ
B00ZS0G0PG
B00WRDS8H0
B00DUGZFWY
B00Y3QOH5G
B06XD5YCKX
B018Y23P7K
B00F5CKWBA,B00KPQCWAU
B00DU15MU4
B00DIHVMEA,B00EZ1ZTV0
B006BGZJJ4
B00QAVNWSK



```
asins_count_ix = data["asins"].value_counts().index
plt.subplots(2,1,figsize=(16,12))
plt.subplot(2,1,1)
data["asins"].value_counts().plot(kind="bar", title="ASIN Frequency",color=['yellow', 'red', 'blue'])
plt.subplot(2,1,2)
sns.pointplot(x="asins", y="reviews.rating", order=asins_count_ix, data=data )
plt.xticks(rotation=90)
plt.show()
```

The most frequently reviewed products have their average review ratings in the 4.5 - 4.8 range, with little variance

Although there is a slight inverse relationship between the ASINs frequency level and average review ratings for the first 4 ASINs, this relationship is not significant since the average review for the first 4 ASINs are rated between 4.5 - 4.8, which is considered good overall reviews

The length of the vertical lines shows that for ASINs with lower frequencies as seen on the bar graph (top), their corresponding average review ratings on the point-plot graph (bottom) have much more variation. As a consequence of the huge variance, we believe that the average review ratings for ASINs with lower frequencies are not important for our research.

However, we believe that due to lesser quality items, as seen by their lower frequencies for ASINs with lower frequencies.

Furthermore, due to their substantially lower frequencies, the final four ASINs have no variation, and even if the review ratings are a perfect 5.0, we should not consider the relevance of these review ratings due to the lower frequency as indicated in the third point.

Chapter 5

Feature Engineering and Selection.

The act of choosing, altering, and converting raw data into features that may be utilized in supervised learning is known as feature engineering. It may be essential to build and train better features in order for machine learning to perform effectively on new jobs. A "feature" is any quantifiable input that may be employed in a predictive manner, as you may know.

We used *porterstemmer*() to stem the words in the review and nltk modules to import stop - words and delete them from the reviews in this project's feature engineering area.

Finally, there's the Bag of Words model, which is divided into two classes. Count Create feature vectors for each review using the Vectorizer() and TF-IDF Vectorizer() classes.

Stemming

a) In the first step we imported all the packages needed and with this line " review = re.sub('[^a-zA-Z]', ' ', str(data['reviews.text'][i])) " we replaced all the characters other than a-z and A-Z with empty character(" ")

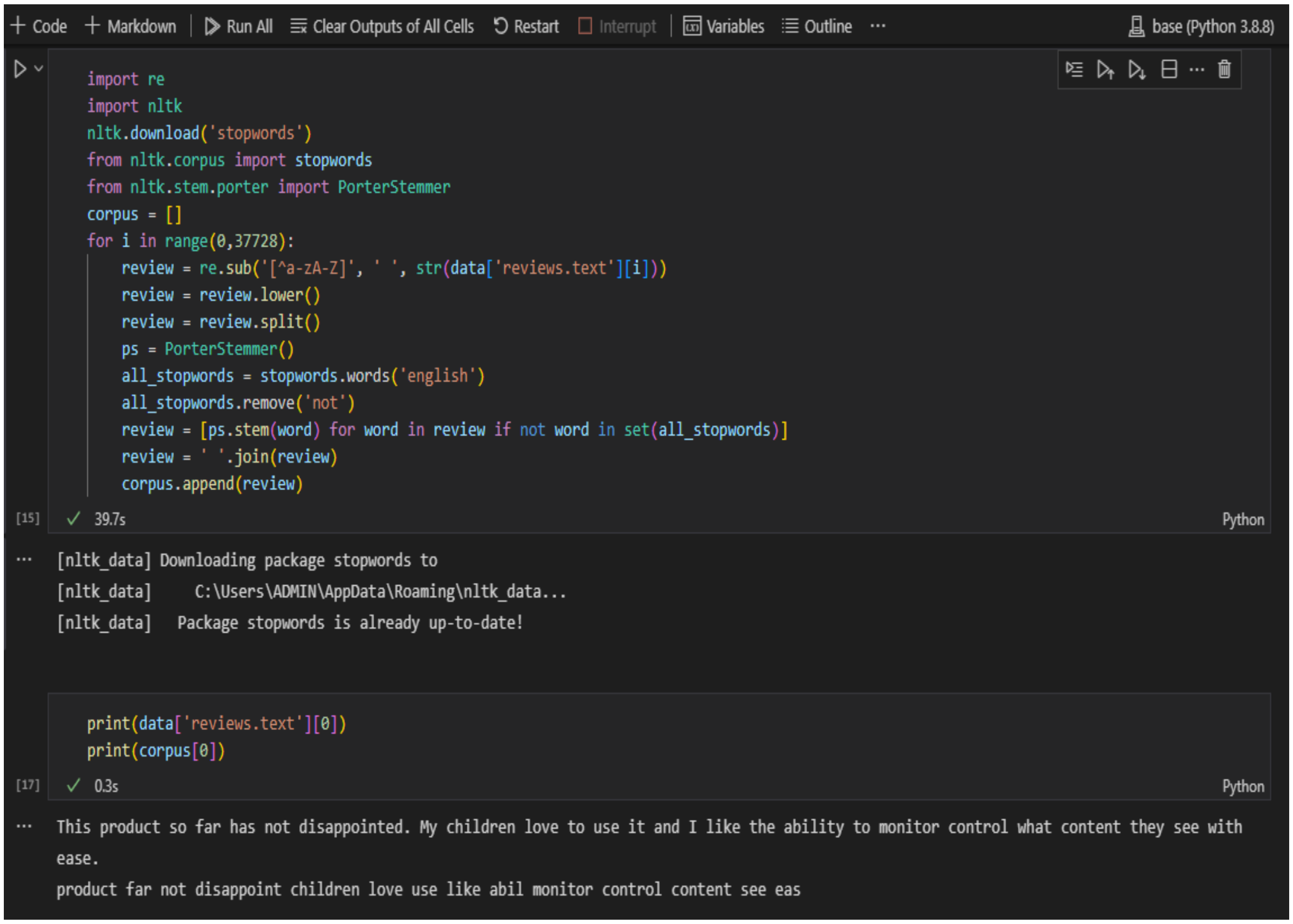
b) In the second line we converted all the uppercase letters to lowercase letters

c) We then split the review based on empty character (" ") to form a list of all the words

d) In the next step we stem the each word in the review and also immediately after stemming if that word is in the stop words then remove that word and add it to the list

e) We then again join all the words in the list to again form the review

f) Finally this newly formed string is added to corpus list and the same procedure is repeated for all the reviews


+ Code + Markdown | Run All | Clear Outputs of All Cells | Restart | Interrupt | Variables | Outline ...
base (Python 3.8.8)

```
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
corpus = []
for i in range(0,37728):
    review = re.sub('[^a-zA-Z]', ' ', str(data['reviews.text'][i]))
    review = review.lower()
    review = review.split()
    ps = PorterStemmer()
    all_stopwords = stopwords.words('english')
    all_stopwords.remove('not')
    review = [ps.stem(word) for word in review if not word in set(all_stopwords)]
    review = ' '.join(review)
    corpus.append(review)
```

[15] ✓ 39.7s Python

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
print(data['reviews.text'][0])
print(corpus[0])
```

[17] ✓ 0.3s Python

```
This product so far has not disappointed. My children love to use it and I like the ability to monitor control what content they see with
ease.
product far not disappoint children love use like abil monitor control content see eas
```

In the above figure we printed the sample output of the first reviews text before and after stemming we can see that all the unnecessary stop-words are removed and the words like “ability” is stemmed to ability -> abil , the same happens for all the other reviews and all the reviews length is reduced and this helps in the next step (Bag of Words) so that the length of the feature vector is reduced and the computation and accuracy is increased.

Bag of Words strategy:

- **Assign fixed integer id to each word occurrence (integer indices to word occurrence dictionary)**
- **X[i,j] where i is the integer indices, j is the word occurrence, and X is an array of words (our training set)**

For sentiment analysis we use the **Bag of Words** model. Let us look briefly about this model

What is a Bag-of-Words?

A bag-of-words is a text representation that shows where words appear in a document. There are two aspects to it:

A list of terms that are familiar to you. A metric for counting the number of words that have been heard before. Any information about the sequence or structure of words in the text is destroyed, which is why it's dubbed a "bag" of words. The model simply cares if identified terms appear in the document, not where in the document they appear.

The bag-of-words technique is a frequently used element in sentence and document extraction procedures (BOW). We examine the histogram of the words inside the text in this technique, treating each word count as a feature.

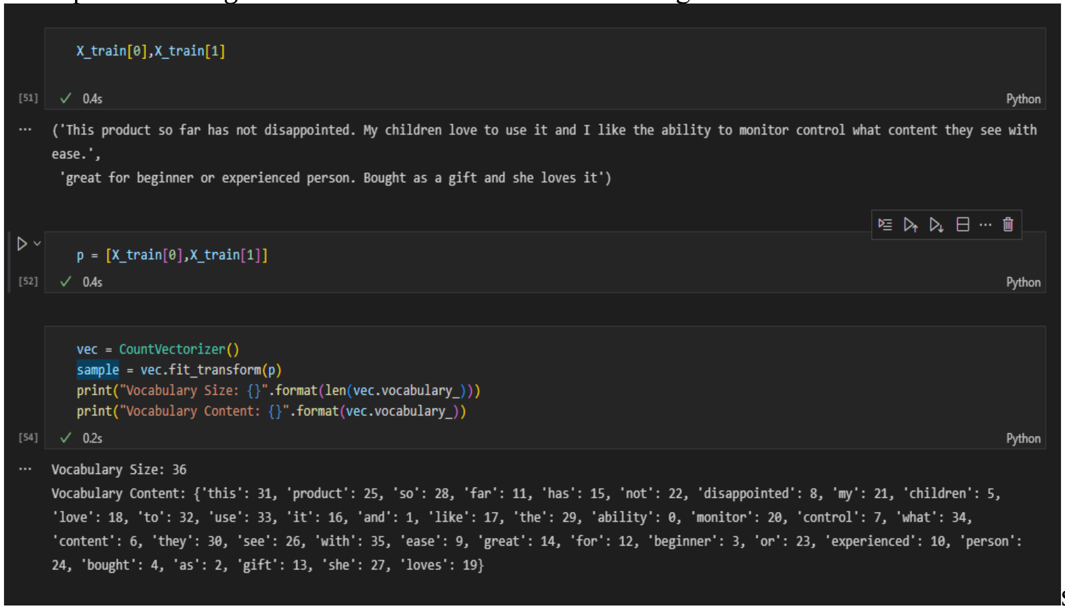
When papers have similar content, it's assumed that they're similar. Furthermore,

The assumption is that documents with comparable content are similar. Furthermore, we may deduce something about the document's significance only from its content. You may make the bag-of-words as basic or as complex as you like. The difficulty arises from determining how to create a vocabulary of known words (or tokens) as well as how to rate the existence of known terms.

Because we can't train models on string data, we used a count vectorizer to turn the text inputs of reviews into numerical feature vectors.

Count-Vectorizer

Example of the Bag-of-Words Model : Let's make the bag-of-words model concrete with two review



```
X_train[0],X_train[1]
```

```
('This product so far has not disappointed. My children love to use it and I like the ability to monitor control what content they see with ease.',
 'great for beginner or experienced person. Bought as a gift and she loves it')
```

```
p = [X_train[0],X_train[1]]
```

```
vec = CountVectorizer()
sample = vec.fit_transform(p)
print("Vocabulary Size: {}".format(len(vec.vocabulary_)))
print("Vocabulary Content: {}".format(vec.vocabulary_))
```

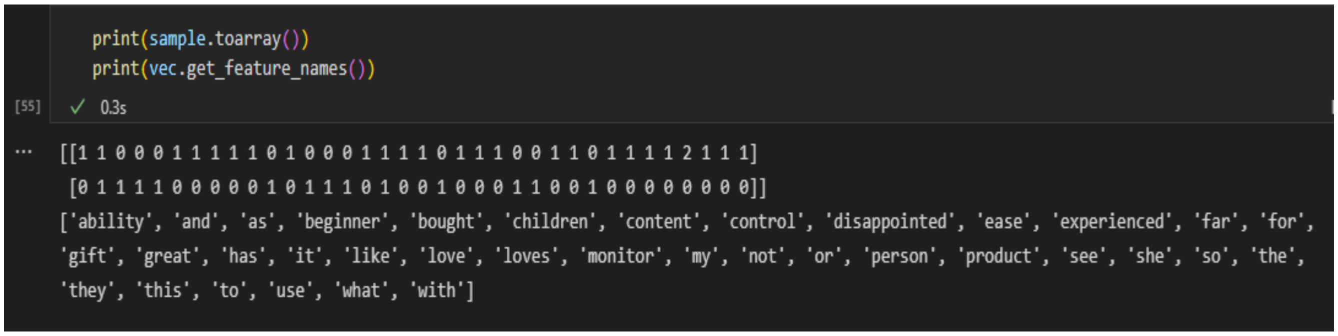
```
Vocabulary Size: 36
Vocabulary Content: {'this': 31, 'product': 25, 'so': 28, 'far': 11, 'has': 15, 'not': 22, 'disappointed': 8, 'my': 21, 'children': 5,
'love': 18, 'to': 32, 'use': 33, 'it': 16, 'and': 1, 'like': 17, 'the': 29, 'ability': 0, 'monitor': 20, 'control': 7, 'what': 34,
'content': 6, 'they': 30, 'see': 26, 'with': 35, 'ease': 9, 'great': 14, 'for': 12, 'beginner': 3, 'or': 23, 'experienced': 10, 'person':
24, 'bought': 4, 'as': 2, 'gift': 13, 'she': 27, 'loves': 19}
```

s

The length of the feature vector is 36 since the vocabulary size (number of unique words) in the two reviews is 36.

I've displayed the two feature vectors of two reviews, together with their index words, in the image below. To summarize, the 0th index represents the word "ability," the 1st index represents the word "and," and so on...

There are 36 words for each of the 36 indices.



```
print(sample.toarray())
print(vec.get_feature_names())
```

```
[55]  ✓ 0.3s
```

```
[[1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 1 1 0 1 1 1 0 0 1 1 0 1 1 1 1 2 1 1 1]
 [0 1 1 1 1 0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0]]
['ability', 'and', 'as', 'beginner', 'bought', 'children', 'content', 'control', 'disappointed', 'ease', 'experienced', 'far', 'for',
'gift', 'great', 'has', 'it', 'like', 'love', 'loves', 'monitor', 'my', 'not', 'or', 'person', 'product', 'see', 'she', 'so', 'the',
'they', 'this', 'to', 'use', 'what', 'with']
```

The values of the index in each array is how many times the that particular word is repeated in that review from the first feature vector the 31st word represents the word “to” as you can see in the first review the word “to” is repeated twice so the value in the array is “2”, the same for all the other words in the array

TF-IDF

Term Frequency – Inverse Document Frequency, or TF-IDF for short, is a scoring method in which:

Term Frequency: This is a score based on how often a term appears in the current document.

Inverse Document Frequency (IDF) is a scoring system that determines how uncommon a word is across texts (y=unique).

The results are based on a weighted average, which means that not all words are equally relevant or intriguing. The ratings have the effect of emphasizing words in a document that are unique (contain useful information). As a result, the IDF of a rare term is likely to be high, whereas the IDF of a common term is likely to be low.

In the example below, a term frequency and inverse term frequency score are generated for each array for index, and the feature vector is modified.

$$TF(i,j)=n(i,j)/\Sigma\ n(i,j)$$

Where,

$n(i,j)$ = number of times nth word occurred in a document

$\Sigma n(i,j)$ = total number of words in a document.

$$IDF=1+\log(N/dN)$$

Where,

N=Total number of documents in the dataset

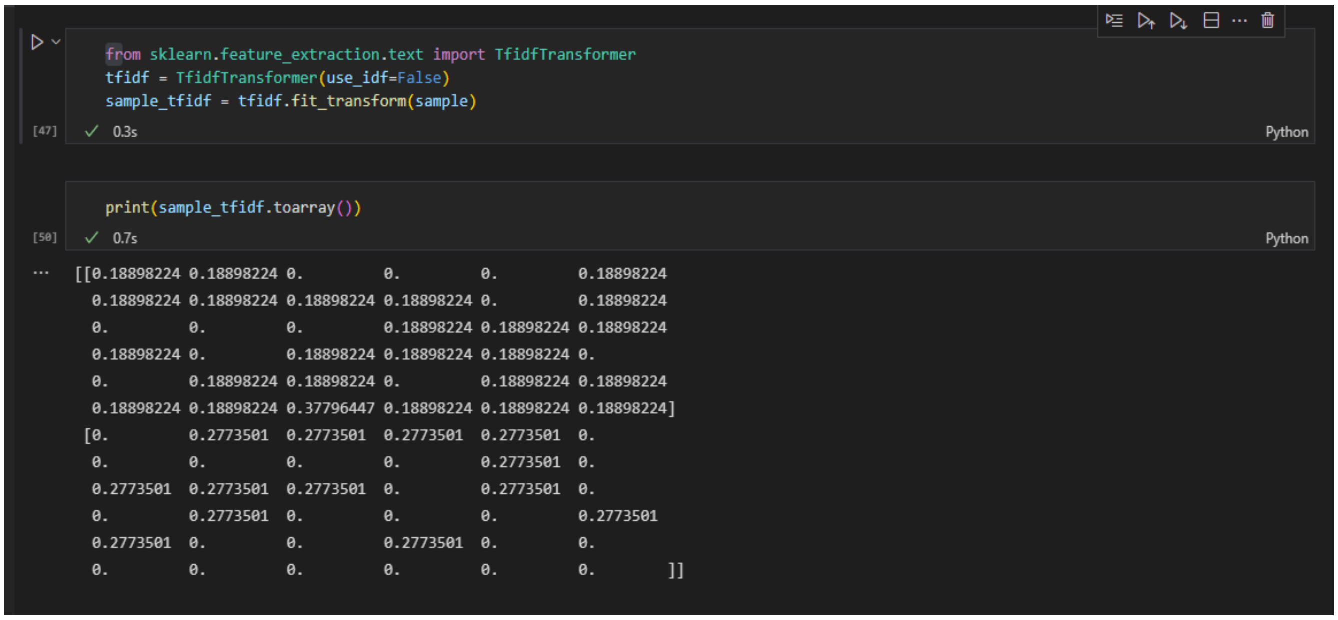
dN=total number of documents in which nth word occur

Also, note that the 1 added in the above formula is so that terms with zero IDF don't get suppressed entirely. This process is known as IDF smoothing.

The TF-IDF is obtained by

$$TF\text{-}IDF = TF*IDF$$

Example for the two reviews is shown below and you can observe the term frequencies in the vector



```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf = TfidfTransformer(use_idf=False)
sample_tfidf = tfidf.fit_transform(sample)
```

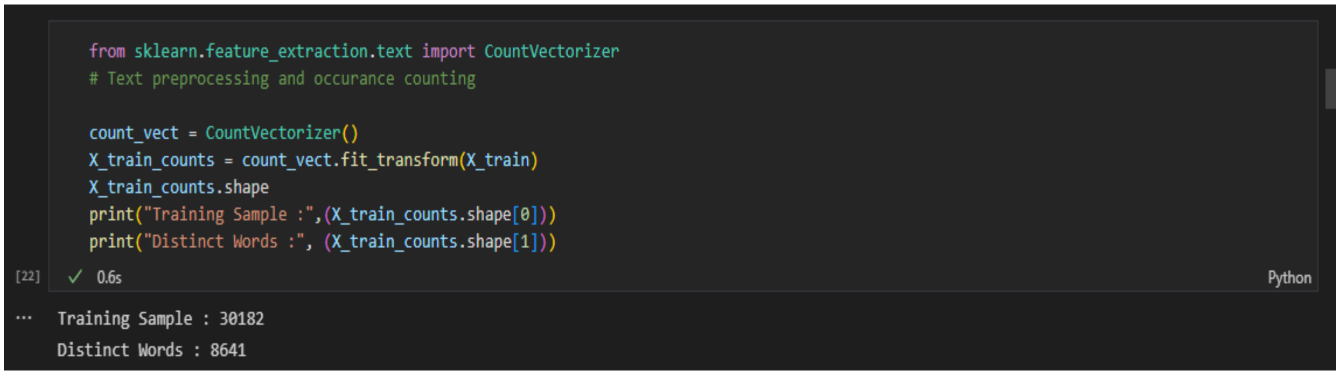
[47] 0.3s

```
print(sample_tfidf.toarray())
```

[50] 0.7s

```
[[0.18898224 0.18898224 0.         0.         0.         0.18898224
  0.18898224 0.18898224 0.18898224 0.18898224 0.         0.18898224
  0.         0.         0.         0.18898224 0.18898224 0.18898224
  0.18898224 0.         0.18898224 0.18898224 0.18898224 0.
  0.         0.18898224 0.18898224 0.         0.18898224 0.18898224
  0.18898224 0.18898224 0.37796447 0.18898224 0.18898224 0.18898224]
 [0.         0.2773501  0.2773501  0.2773501  0.2773501  0.
  0.         0.         0.         0.         0.2773501  0.
  0.2773501  0.2773501  0.2773501  0.         0.2773501  0.
  0.         0.2773501  0.         0.         0.         0.2773501
  0.2773501  0.         0.         0.2773501  0.         0.
  0.         0.         0.         0.         0.         0.        ]]
```

As you seen this example for two reviews the same process is done for all the reviews you can see the vocabulary size or the length of the unique features is ‘8641’ and so each array is of length 8641 and the model is trained for total rows(data points) of ‘30182’ [Xtrain length]



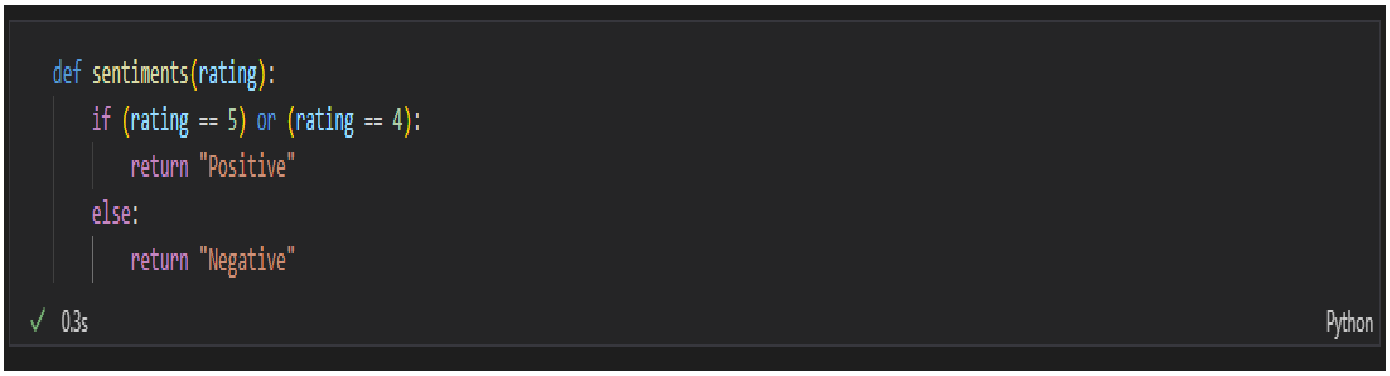
```
from sklearn.feature_extraction.text import CountVectorizer
# Text preprocessing and occurance counting

count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)
X_train_counts.shape
print("Training Sample :",(X_train_counts.shape[0]))
print("Distinct Words :", (X_train_counts.shape[1]))
```

```
[22]  ✓ 0.6s                                                    Python

... Training Sample : 30182
    Distinct Words : 8641
```

Chapter 6

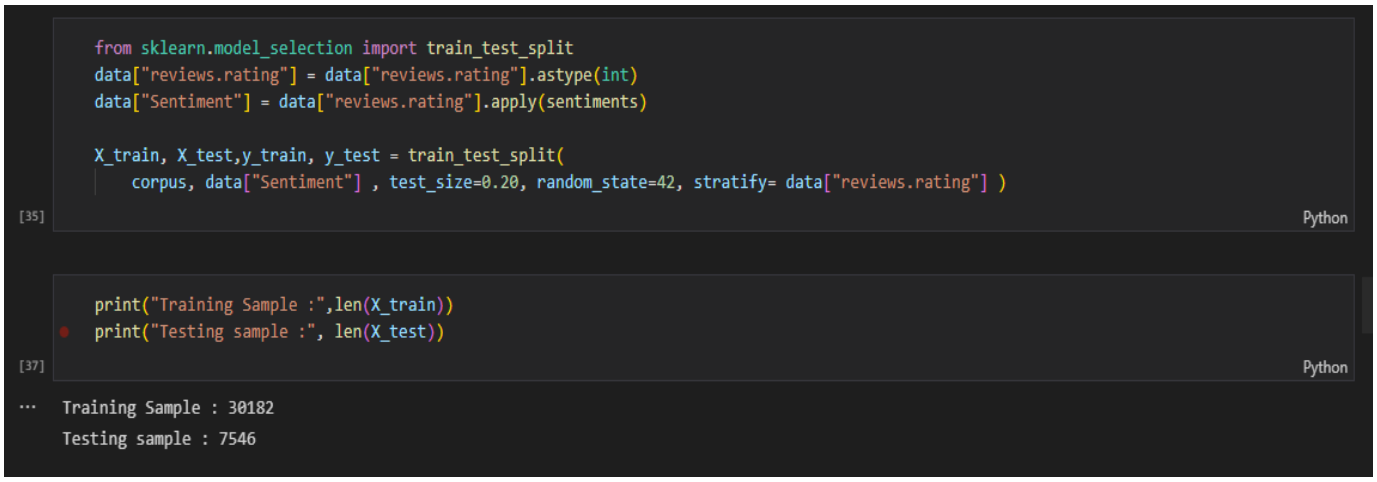
Splitting Dataset into Train and Test Set

Before dividing the data, we must first define the target variable. Here, we must determine if the review is good or negative, thus we divide the reviews into positive (with a rating of 4 or 5) and negative (with any other rating).we used train_test_split function on corpus.The parameter stratify makes sure that the data split is even ie both the train and test data set contains atleast 1 or more data points with all the included review ratings[1-5]



```
def sentiments(rating):
    if (rating == 5) or (rating == 4):
        return "Positive"
    else:
        return "Negative"
```

```
✓ 0.3s                                                          Python
```



```
from sklearn.model_selection import train_test_split
data["reviews.rating"] = data["reviews.rating"].astype(int)
data["Sentiment"] = data["reviews.rating"].apply(sentiments)

X_train, X_test,y_train, y_test = train_test_split(
    corpus, data["Sentiment"] , test_size=0.20, random_state=42, stratify= data["reviews.rating"] )
```

[35] Python

```
print("Training Sample :",len(X_train))
print("Testing sample :", len(X_test))
```

[37] Python

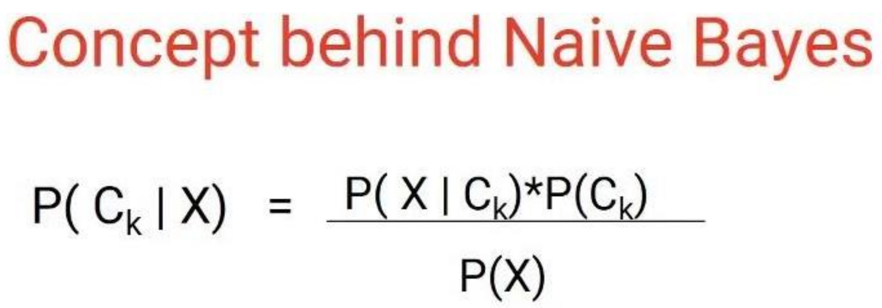
```
Training Sample : 30182
Testing sample : 7546
```

Chapter 7

Models

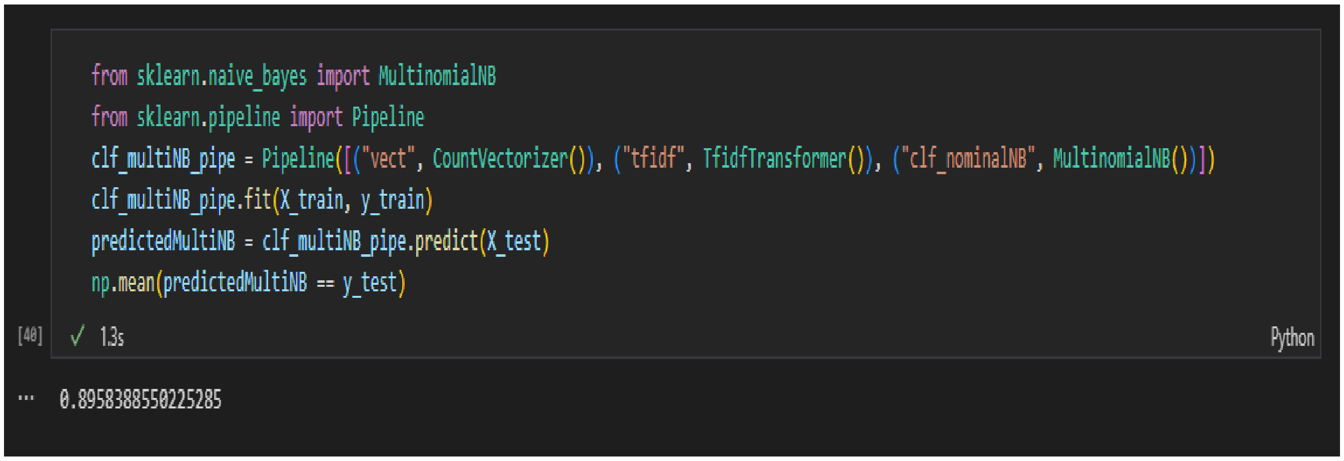
Multinomial Naive Bayes

- Multinomial While disregarding non-occurrences of a feature, Naive Bayes is best suited for word counts, where data is commonly represented as word vector counts (number of times outcome number X[i,j] is seen throughout the n trials).
- P(x|y), where x is the feature and y is the classifier, is a simplified form of Bayes Theorem in which all features are supposed to be conditioned independently of each other (the classifiers).



Concept behind Naive Bayes

$$P(C_k \mid X) = \frac{P(X \mid C_k) * P(C_k)}{P(X)}$$



```
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
clf_multiNB_pipe = Pipeline([("vect", CountVectorizer()), ("tfidf", TfidfTransformer()), ("clf_nominalNB", MultinomialNB())])
clf_multiNB_pipe.fit(X_train, y_train)
predictedMultiNB = clf_multiNB_pipe.predict(X_test)
np.mean(predictedMultiNB == y_test)
```

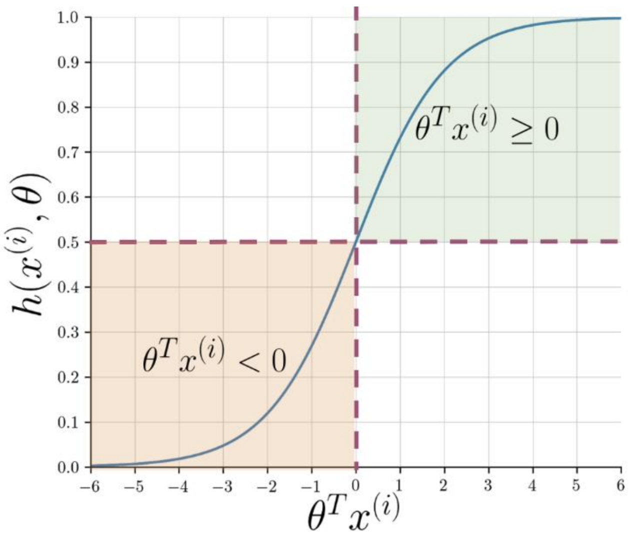
[40] ✓ 1.3s Python

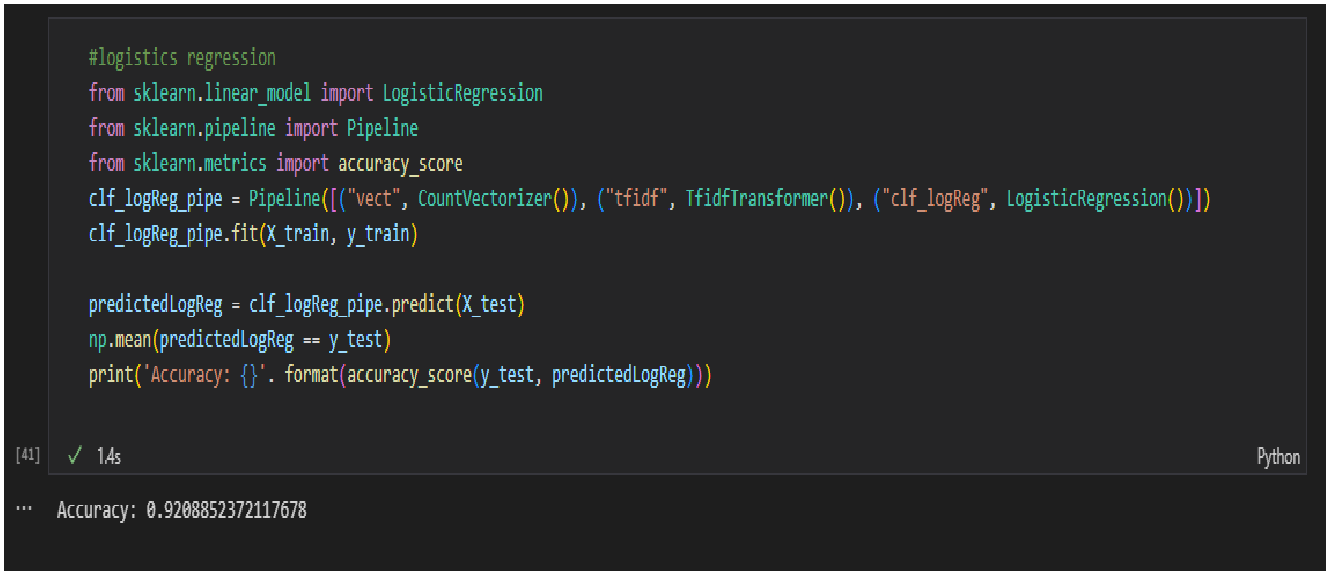
```
0.8958388550225285
```

Logistic Regression Classifier

When the dependent variable is dichotomous, logistic regression is the best regression strategy to use (binary). To describe data and explain the connection between one dependent binary variable and one or more nominal, ordinal, interval, or ratio-level independent variables, logistic regression is utilized for work.

$$h(x^{(i)}, \theta) = \frac{1}{1 + e^{-\theta^T x^{(i)}}}$$





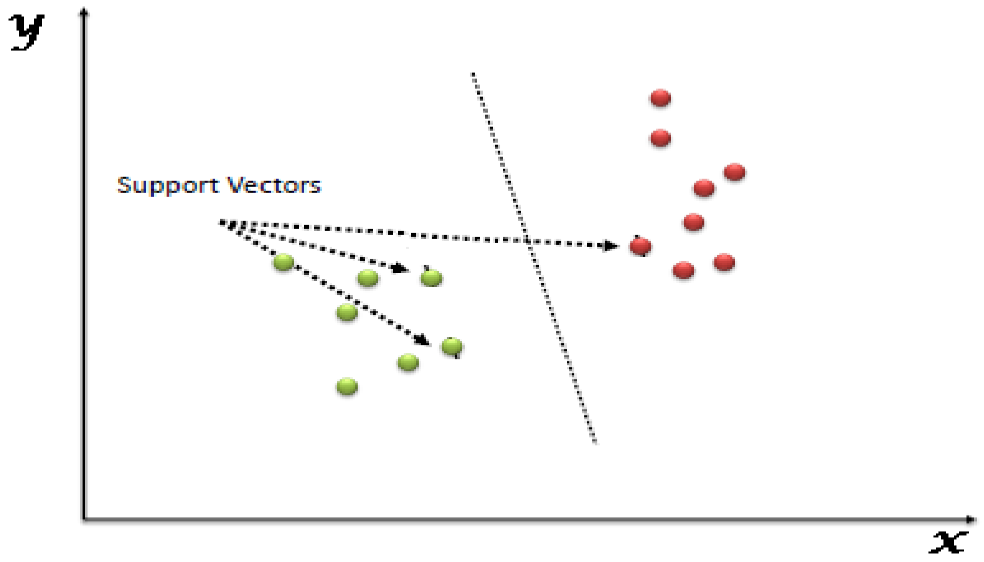
```
#logistics regression
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
clf_logReg_pipe = Pipeline([("vect", CountVectorizer()), ("tfidf", TfidfTransformer()), ("clf_logReg", LogisticRegression())])
clf_logReg_pipe.fit(X_train, y_train)

predictedLogReg = clf_logReg_pipe.predict(X_test)
np.mean(predictedLogReg == y_test)
print('Accuracy: {}'. format(accuracy_score(y_test, predictedLogReg)))
```

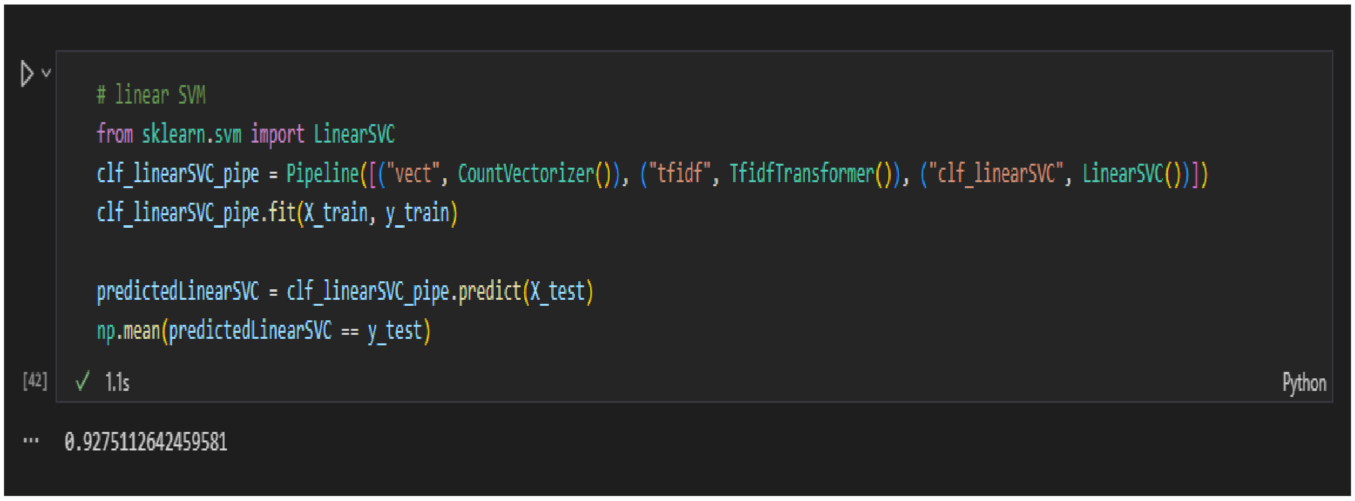
[41] ✓ 1.4s Python

```
Accuracy: 0.9208852372117678
```

Support Vector Machine Classifier

The "Support Vector Machine" (SVM) is a supervised machine learning technique that can solve classification and regression problems. Each data item is plotted as a point in n-dimensional space (where n is the number of features you have), with the value of each feature being the value of a certain coordinate in the SVM algorithm. Then we accomplish classification by locating the hyper-plane that clearly distinguishes the two classes.




```
# linear SVM
from sklearn.svm import LinearSVC
clf_linearSVC_pipe = Pipeline([("vect", CountVectorizer()), ("tfidf", TfidfTransformer()), ("clf_linearSVC", LinearSVC())])
clf_linearSVC_pipe.fit(X_train, y_train)

predictedLinearSVC = clf_linearSVC_pipe.predict(X_test)
np.mean(predictedLinearSVC == y_test)
```
[42] ✓ 1.1s Python

```
0.9275112642459581
```

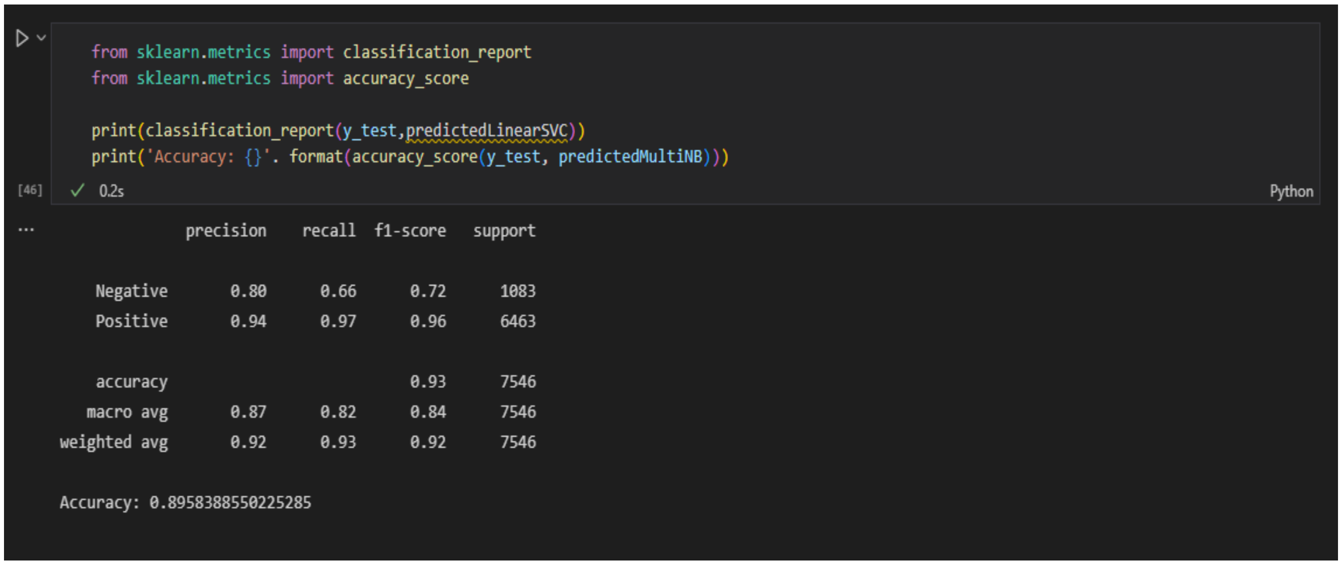
Looks like all the models performed well, but we will use the Support Vector Machine Classifier since it has the highest accuracy level at 92.75%.

Classification report

- Precision identifies how many of the things chosen were right.
- Recall: indicates how many of the things that should have been chosen were really chosen.

- The recall and accuracy weights are measured by the F1 score (1 means precision and recall are equally important, 0 otherwise)
- The number of instances of each class is known as support.

a)Multinomial Naïve Bayes

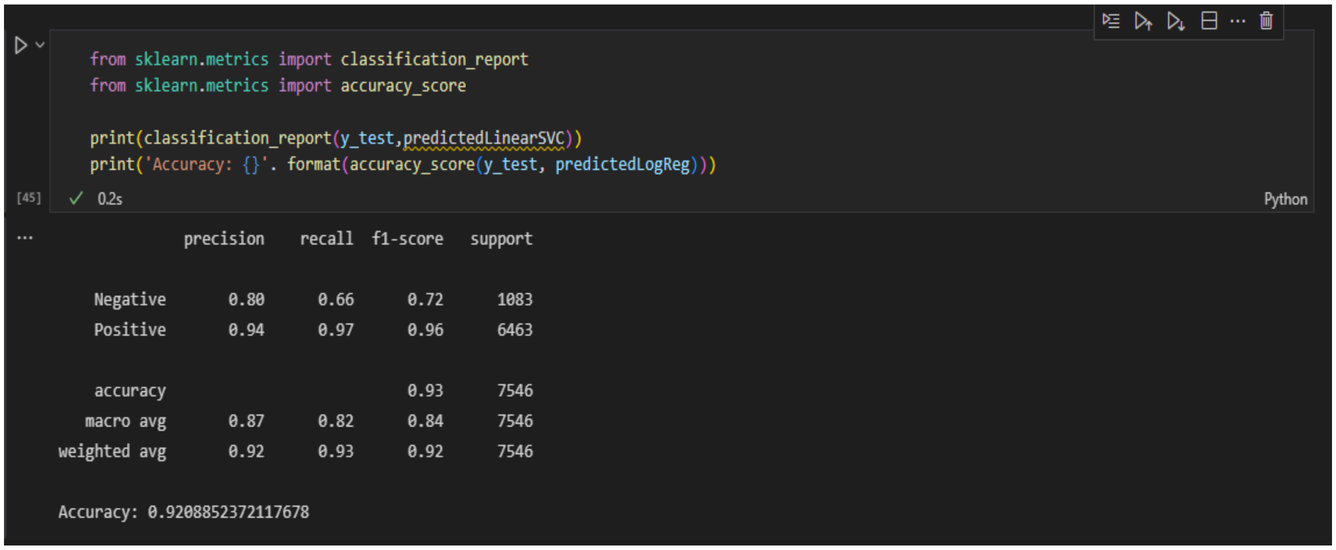


```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

print(classification_report(y_test,predictedLinearSVC))
print('Accuracy: {}'. format(accuracy_score(y_test, predictedMultiNB)))
```

[46] ✓ 0.2s Python

```
              precision    recall  f1-score   support

    Negative       0.80      0.66      0.72      1083
    Positive       0.94      0.97      0.96      6463

    accuracy                           0.93      7546
   macro avg       0.87      0.82      0.84      7546
weighted avg       0.92      0.93      0.92      7546

Accuracy: 0.8958388550225285
```

b)Logistic Regression

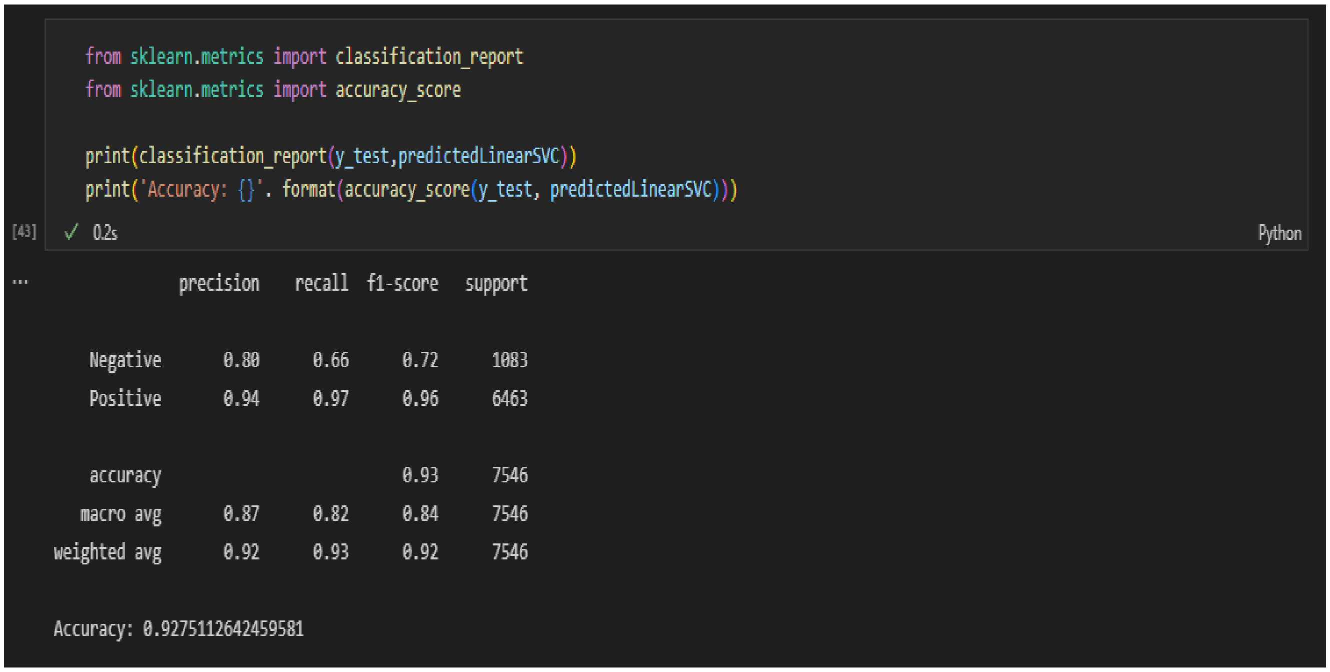


```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

print(classification_report(y_test,predictedLinearSVC))
print('Accuracy: {}'. format(accuracy_score(y_test, predictedLogReg)))
```

[45] ✓ 0.2s Python

```
              precision    recall  f1-score   support

    Negative       0.80      0.66      0.72      1083
    Positive       0.94      0.97      0.96      6463

    accuracy                           0.93      7546
   macro avg       0.87      0.82      0.84      7546
weighted avg       0.92      0.93      0.92      7546

Accuracy: 0.9208852372117678
```

c) Support Vector Machines



```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

print(classification_report(y_test,predictedLinearSVC))
print('Accuracy: {}'. format(accuracy_score(y_test, predictedLinearSVC)))
```

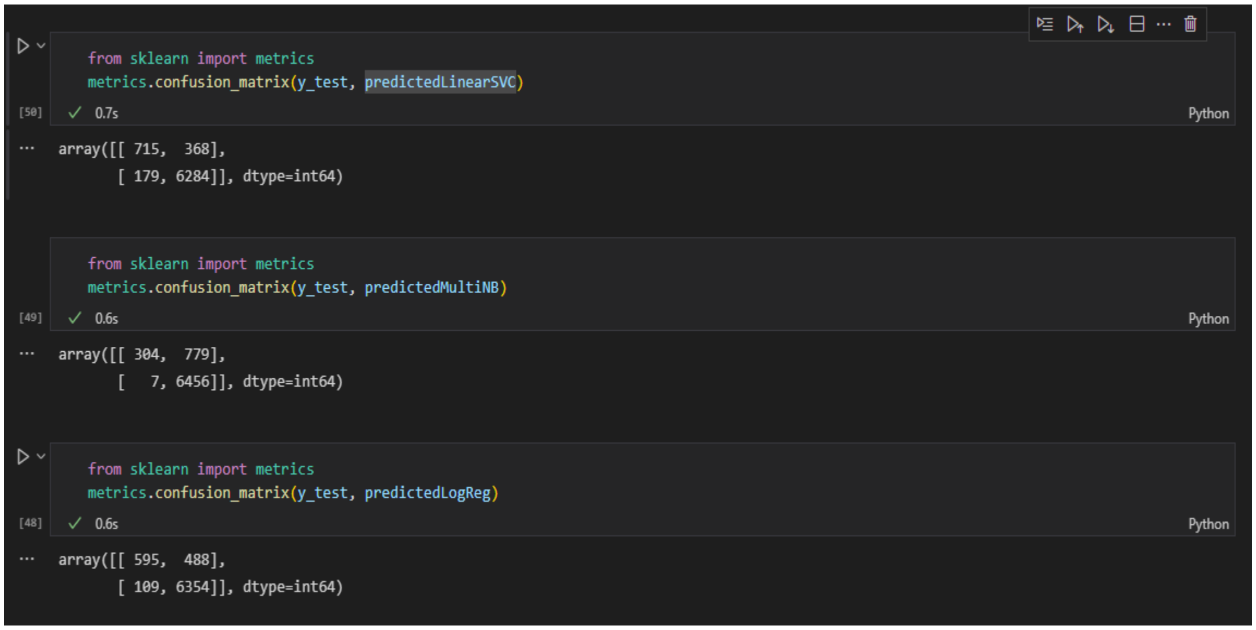
[43] ✓ 0.2s Python

```
              precision    recall  f1-score   support

    Negative       0.80      0.66      0.72      1083
    Positive       0.94      0.97      0.96      6463

    accuracy                           0.93      7546
   macro avg       0.87      0.82      0.84      7546
weighted avg       0.92      0.93      0.92      7546

Accuracy: 0.9275112642459581
```

Chapter 8

Confusion matrix

The confusion matrix is a N x N matrix that is used to evaluate the performance of a classification model, with N being the number of target classes. The matrix compares the actual goal values to the machine learning model's predictions. This provides us with a comprehensive picture of how well our classification model is working and the types of errors it makes.



```
[50]
from sklearn import metrics
metrics.confusion_matrix(y_test, predictedLinearSVC)
✓ 0.7s                                                    Python

array([[ 715,  368],
       [ 179, 6284]], dtype=int64)
```

```
[49]
from sklearn import metrics
metrics.confusion_matrix(y_test, predictedMultiNB)
✓ 0.6s                                                    Python

array([[ 304,  779],
       [   7, 6456]], dtype=int64)
```

```
[48]
from sklearn import metrics
metrics.confusion_matrix(y_test, predictedLogReg)
✓ 0.6s                                                    Python

array([[ 595,  488],
       [ 109, 6354]], dtype=int64)
```

Chapter 9

Conclusion & Future Scope

The findings of this study back with our prior data exploration findings, which found that the data is heavily skewed toward positive evaluations, as seen by the lower support numbers in the categorization report. Furthermore, both positive and negative evaluations have a high standard deviation with low frequencies, which we do not deem relevant, as seen by the lower accuracy, recall, and F1 scores in the categorization report.

Despite the fact that the Neutral and Negative findings are not particularly good predictors in this data set, the sentiment analysis is predicted with a 92.23 percent accuracy rate. As a result, we feel at ease with the skewed data set. Additionally, when we continue to input additional datasets that are more balanced in the future, this model will re-adjust to a more balanced state.

Note : That the first row will be disregarded because we replaced all NAN with " " earlier. When we originally imported the raw data, we tried to eliminate this row, however Pandas Dataframe didn't like it when we tried to drop all NANs (before stratifying and splitting the dataset). As a consequence, the best remedy was to replace the NAN with " ", and the first row would be omitted in this analysis.
Finally, the overall result indicates that the products in this dataset are typically well-received.

From the analysis above in the classification report, we can see that products with lower reviews are not significant enough to predict these lower rated products are inferior. On the other hand, products that are highly rated are considered superior products, which also perform well and should continue to sell at a high level.
As a result, we need to input more data in order to consider the significance of lower rated products, in order to determine which products should be dropped from Amazon's product roster.
The good news is that despite the skewed dataset, we were still able to build a robust Sentiment Analysis machine learning system to determine if the reviews are positive or negative. This is possible as the machine learning system was able to learn from all the positive, neutral and negative reviews, and fine tune the algorithm in order to avoid biased sentiments.
In future we will try to include more robust algorithms and also attempt to integrate it into a web app so that it can become user friendly.

In conclusion, we were able to correctly link positive, neutral, and negative feelings for each product in Amazon's Catalog, despite the fact that we needed additional data to balance out the lower rated goods in order to weight their relevance.

Chapter 10

References

1. https://www.analyticsvidhya.com/blog/2020/11/create-a-pipeline-to-perform-sentiment-analysis-using-nlp/
2. https://www.researchgate.net/profile/Doaa-Mohey-El-Din/publication/301683673_Paper_34-Enhancement_Bag_of_Words_Model_for_Solving/data/5721dde708aea0447bce3027/Paper-34-Enhancement-Bag-of-Words-Model-for-Solving.pdf
3. https://www.cc.gatech.edu/home/isbell/classes/reading/papers/Rish.pd
4. https://www.sciencedirect.com/science/article/pii/S1877050918308512
5. https://search.proquest.com/openview/e7d1fec1f70f61d4f2fea58304446c3c/1?pq-origsite=gscholar&cbl=5444811