FLIP ROBO

RATING CLASSIFICATION

Submitted by:

BALPREET

ACKNOWLEDGMENT

This includes mentioning of all the references, research papers, data sources, professionals and other resources that helped you and guided you in completion of the project.

Various online sources were used in order to get completed this project such as Analytica Vidhya, Medium, Towards Data Science and Data Trained tutorial and notes and sample of projects done in Github such as:

- https://towardsdatascience.com/text-processing-in-python-29e86ea4114c
- https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908
- https://realpython.com/nltk-nlp-python/
- https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/
- https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk
- https://towardsai.net/p/nlp/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0
- https://towardsdatascience.com/word-embedding-techniques-word2vec-and-tf-idf-explained-c5d02e34d08
- https://www.analyticsvidhya.com/blog/2021/07/bag-of-words-vs-tfidf-vectorization-a-hands-on-tutorial/
- https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/
- https://matheo.uliege.be/bitstream/2268.2/2707/4/Memoire_MarieMartin_s112740.pdf
- https://vuir.vu.edu.au/41279/1/DU%20Jianhua-thesis.pdf
- https://medium.com/analytics-vidhya/jigsaw-unintended-bias-toxic-comment-classification-fa68fe3a27c8
- https://github.com/napsterninad20/Yelp-Analysis-and-Rating-Prediction/blob/master/Coding%20file.ipynb
- https://github.com/F74046080/Rating-Prediction-with-User-Business-Review/blob/master/rating_prediction_methods.ipynb

- https://github.com/prahasan21/Yelp-review-rating-prediction/blob/master/Predicting_Star_Ratings_from_Reviews.ipynb
- https://github.com/F74046080/Rating-Prediction-with-User-Business-Review/blob/master/rating_prediction_methods.ipynb
- https://medium.com/analytics-vidhya/jigsaw-unintended-bias-toxic-comment-classification-fa68fe3a27c8
- https://github.com/NikhilGohil/Social-Media-and-Data-Mining-Project/blob/master/SMDM%20Project%20Report.pdF
- https://towardsdatascience.com/classifying-toxicity-in-online-comment-forums-end-to-end-project-57720af39d0b
- https://medium.datadriveninvestor.com/toxic-comment-classification-a-kaggle-case-study-a929b37150b
- https://www.researchgate.net/publication/348635009_Detecting_Toxic_Remarks_in_Online_Conversations
- https://www.researchgate.net/publication/349929587_Machine_learning_methods_for_toxic_comment_classification_a_systematic_review
- https://www.researchgate.net/publication/334123568_Toxic_Comment_Classification
- https://www.rsisinternational.org/journals/ijrias/DigitalLibrary/Vol.4&Issue11/142-147.pdf
- https://github.com/tianqwang/Toxic-Comment-Classification-Challenge
- https://github.com/topics/toxic-comment-classification
- https://github.com/IBM/MAX-Toxic-Comment-Classifier
- https://github.com/cjvegi/Toxic-Comment-Challenge

INTRODUCTION

- ## Business Problem Framing

  We have a client that wants to predict ratings for the reviews which were written in the past and for which they don't have any ratings. So, we have to build an application which can predict the rating by seeing the review.

- ## Conceptual Background of the Domain Problem

  Generally speaking, every data scientist has to have a minimum domain expertise that implies knowledge and understanding of the essential aspects of the specific field of online reviews and way of sharing their opinions. In other words, a data scientist must know the stuff he/she is involved with. It is essential in order to make and have a valuable results and conclusions. It is essentially required for data mining and pattern discovery. The data scientist needs to know how the essentials of the specific field works so that the discovery, the patterns and evaluation process is guided by an intuitive knowledge of what actually matters both in terms of inputs and outputs as well as of what makes more sense or not.

- ## Review of Literature

  The distribution of rating scores has been the subject of various research. Among others, Hu, Pavlou and Zhang (2009) attempt to understand and demonstrate the existence of a recurrent Jshaped distribution19 in the majority of Amazon product reviews.

  Other authors (Chevalier and Mayzlin (2006), Kadet (2007)) also analyze the distribution of ratings in online reviews and come to the same conclusion: the resulting data presents an asymmetric bimodal distribution, where reviews are overwhelmingly positive.

  Max Woolf (2017) illustrates and generalizes the J-shaped distribution of online reviews to various Amazon product categories. The figures hereunder show how user ratings are distributed among the reviews.

In order to alleviate the problem of class imbalance in classification, resampling methods aiming at balancing the datasets have been the focus of various research23 . More specifically, Chawla, Bowyer, Hall and Kegelmeyer (2002) concentrate on a novel technique called SMOTE. This resampling technique is by far the most known for its effectiveness and simplicity. It has been extensively studied in the literature giving rise to many variants of the original method. However, we will stick to the initial version of SMOTE in the framework of this dissertation. Numerous experiments including Dal Pozzolo, Caelen and Bontempi (2013) were conducted on a variety of datasets to measure the performance of the different sampling techniques. They show that the SMOTE approach clearly outperforms the other existing resampling techniques (under and oversampling) resulting in better classification performance in most of the cases. Blagus and Lusa (2013) nuance these findings. While SMOTE is effectively beneficial when presented to low-dimensional data, this sampling method is not effective in most circumstances when data are high-dimensional, sometimes showing a worst performance than the random undersampling method.

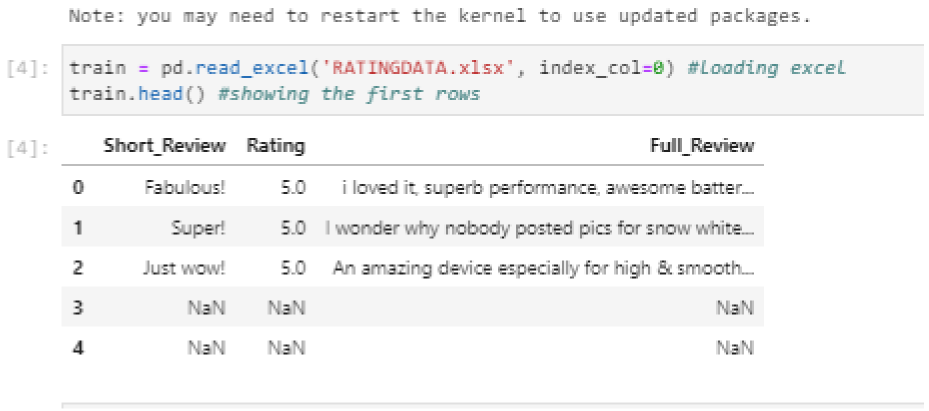
- ## Motivation for the Problem Undertaken

My interest started when the online platform was used and enabled to write, without any obligation to reveal oneself directly, means that this new medium of virtual communication allows people to feel greater freedom in the way they express themselves. My interest started as the Social media have become a ground for heated discussions which may frequently result in the use of insulting and offensive language since the starting the freedom of express in the internet and especially an option to state and opine as anomously.

Analytical Problem Framing

- Data Sources and their formats

What are the data sources, their origins, their formats and other details that you find necessary? They can be described here. Provide a proper data description. You can also add a snapshot of the data.



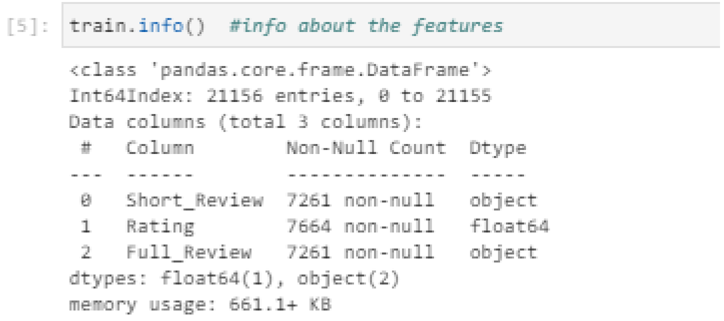
```
Note: you may need to restart the kernel to use updated packages.
```

```
[4]: train = pd.read_excel('RATINGDATA.xlsx', index_col=0) #Loading excel
     train.head() #showing the first rows
```

[4]:

	Short_Review	Rating	Full_Review
0	Fabulous!	5.0	i loved it, superb performance, awesome batter...
1	Super!	5.0	I wonder why nobody posted pics for snow white...
2	Just wow!	5.0	An amazing device especially for high & smooth...
3	NaN	NaN	NaN
4	NaN	NaN	NaN

In the training dataset, we have a Short Review text and a Full Review alongside with a Rating for each combination for short and full review.



```
[5]: train.info()  #info about the features

<class 'pandas.core.frame.DataFrame'>
Int64Index: 21156 entries, 0 to 21155
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Short_Review  7261 non-null   object 
 1   Rating        7664 non-null   float64
 2   Full_Review   7261 non-null   object 
dtypes: float64(1), object(2)
memory usage: 661.1+ KB
```

Notice that the data contains 21156 observations with 3 columns for the training and testing purposes.

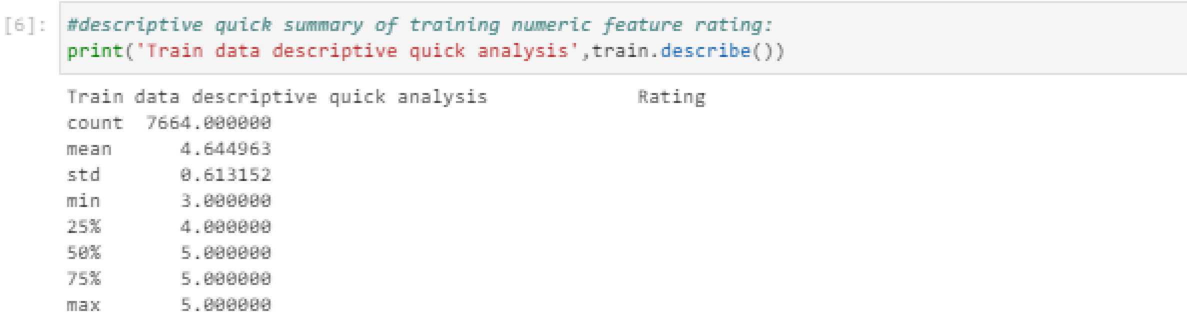
As we see, the Short Review and Full Review are the 2 features whose data type is object.

The remaining one, the target column, is a numerical feature defined as float type.

As seen in the non-null count, we have that all the features including the target feature have missing values as all the features don’t have equal number of instances as the number of cases or entries. The number of non-null rows are less than the total entries of the dataset.

Mathematical/ Analytical Modeling of the Problem

Describe the mathematical, statistical and analytics modelling done during this project along with the proper justification.



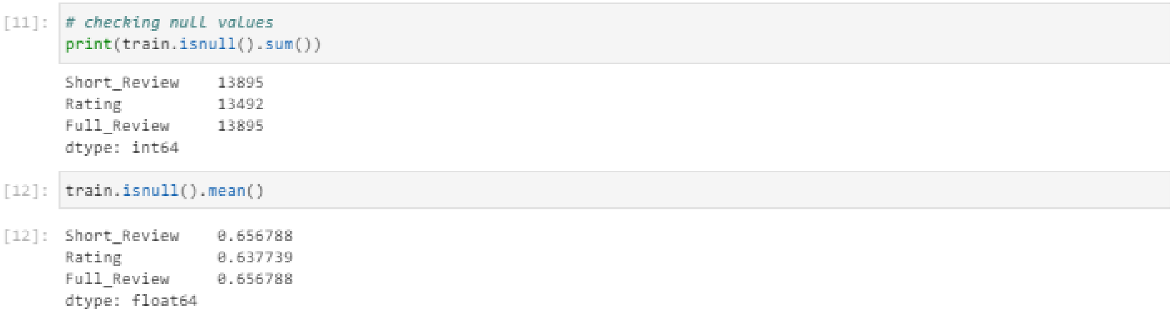
```
[6]: #descriptive quick summary of training numeric feature rating:
     print('Train data descriptive quick analysis',train.describe())
```

```
Train data descriptive quick analysis              Rating
count  7664.000000
mean      4.644963
std       0.613152
min       3.000000
25%       4.000000
50%       5.000000
75%       5.000000
max       5.000000
```

Regarding the feature Rating, we see we have right side skewed distribution, which means our mean is higher than the median.

We also see 75% percentile is same as the max value, which indicates there is NO OUTLIER in our numerical feature.

And as we said, the count of values (7664) of Rating is less than the total number of rows, which means we have NULL Values in our numerical feature RATING.

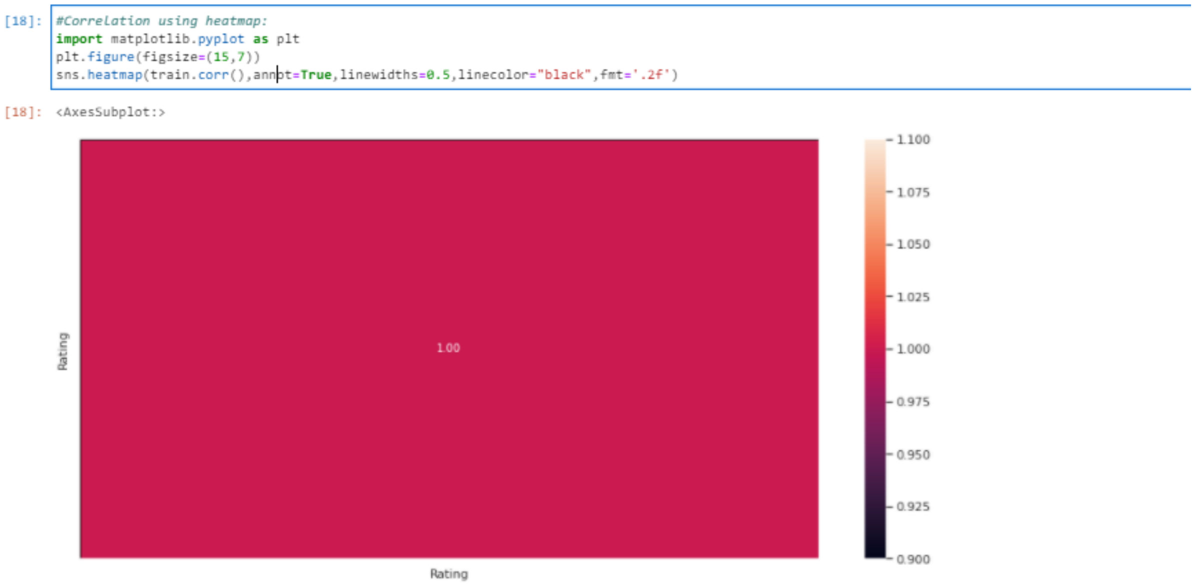


```
[11]: # checking null values
      print(train.isnull().sum())

      Short_Review    13895
      Rating          13492
      Full_Review     13895
      dtype: int64

[12]: train.isnull().mean()

[12]: Short_Review    0.656788
      Rating          0.637739
      Full_Review     0.656788
      dtype: float64
```

OK, checking the null values, we see that we have 65,67% of our Short_Reviews rows are missing, which means no values in there.

Regarding numerical feature Rating, we are missing 63,77% of the rows.

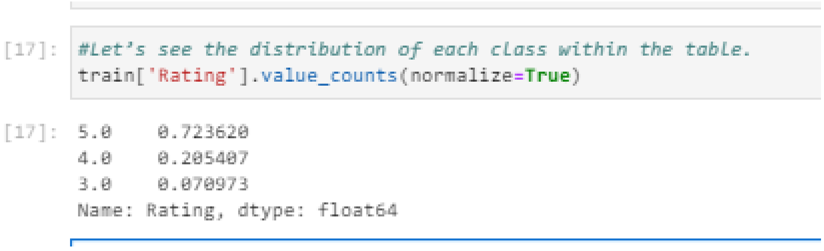
And finally, Full Reviews has also 65,67% of its rows which have no data/reviews in there.



```
[18]: #Correlation using heatmap:
      import matplotlib.pyplot as plt
      plt.figure(figsize=(15,7))
      sns.heatmap(train.corr(),annot=True,linewidths=0.5,linecolor="black",fmt='.2f')

[18]: <AxesSubplot:>
```

No correlation is expected between our features as we only have one numerical feature. No other numerical feature to correlate with.

We have an imbalance in our target feature, so we will re-balance it through the tecnique SMOTE:



```
[17]: #Let's see the distribution of each class within the table.
      train['Rating'].value_counts(normalize=True)

[17]: 5.0    0.723620
      4.0    0.205407
      3.0    0.070973
      Name: Rating, dtype: float64
```

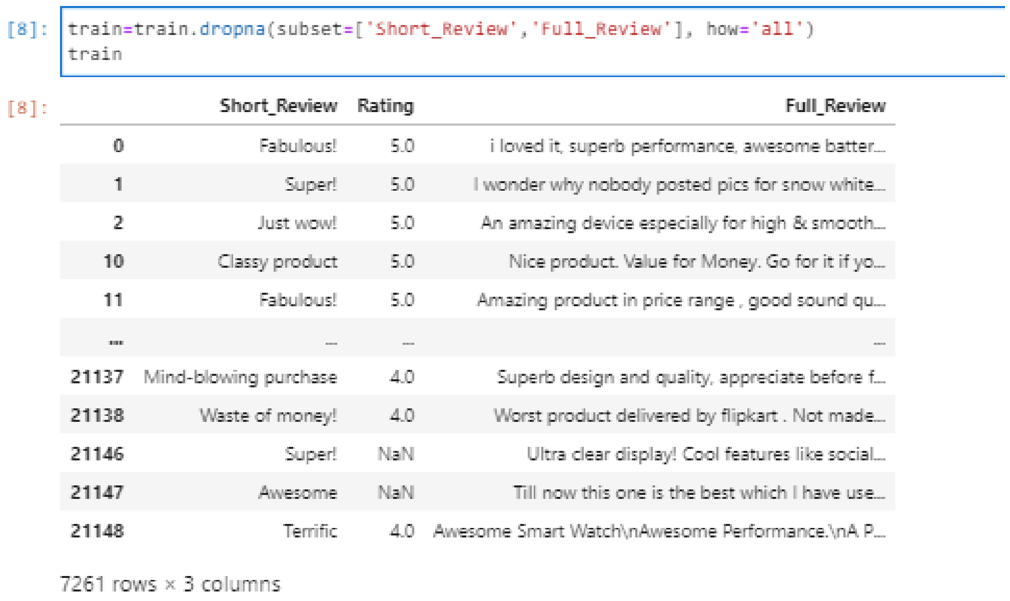
As we just mentioned, our target feature "Rating" is an imbalanced feature which has mostly rating number 5 and which represents 72% of the ratings. As we said, we will use SMOTE techique to adjust and remove this imbalance issue.

• Data Preprocessing Done

What were the steps followed for the cleaning of the data? What were the assumptions done and what were the next actions steps over that?

We preprocessed missing values as follows:

All 3 columns have missing data as commented previously. So, let's drop missing values:



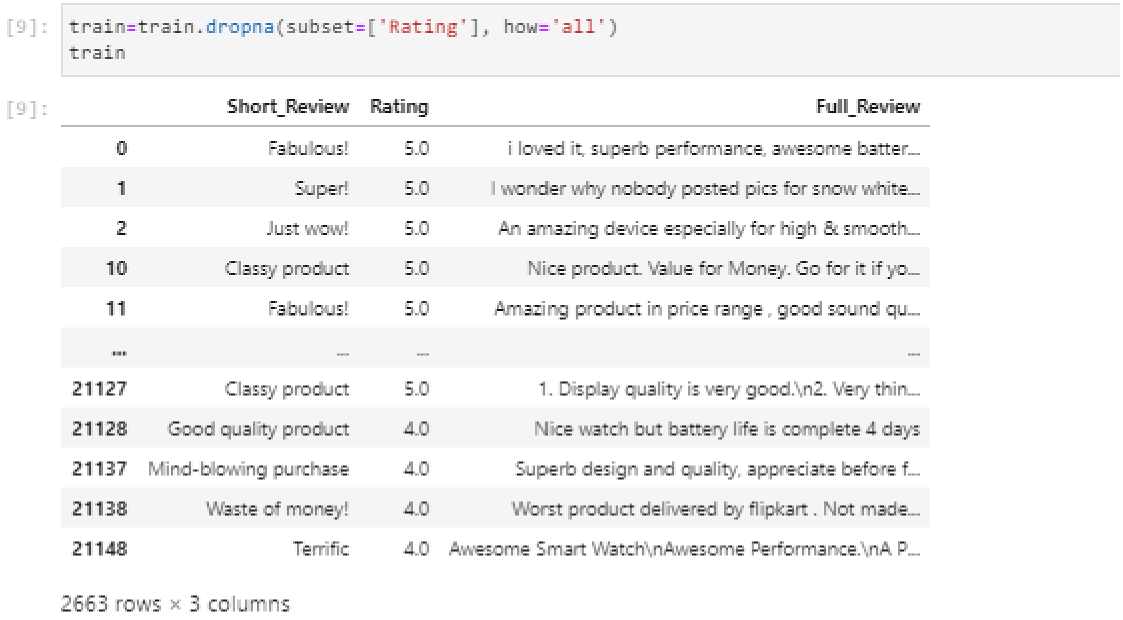
```
[8]: train=train.dropna(subset=['Short_Review','Full_Review'], how='all')
     train
```

[8]:

	Short_Review	Rating	Full_Review
0	Fabulous!	5.0	i loved it, superb performance, awesome batter...
1	Super!	5.0	I wonder why nobody posted pics for snow white...
2	Just wow!	5.0	An amazing device especially for high & smooth...
10	Classy product	5.0	Nice product. Value for Money. Go for it if yo...
11	Fabulous!	5.0	Amazing product in price range , good sound qu...
...	...	...	...
21137	Mind-blowing purchase	4.0	Superb design and quality, appreciate before f...
21138	Waste of money!	4.0	Worst product delivered by flipkart . Not made...
21146	Super!	NaN	Ultra clear display! Cool features like social...
21147	Awesome	NaN	Till now this one is the best which I have use...
21148	Terrific	4.0	Awesome Smart Watch\nAwesome Performance.\nA P...

7261 rows × 3 columns

Dropping first of all 'Short_Review' and 'Full_Review' Nan values when both have no values in it as we can't map the rating with any comment.



```
[9]: train=train.dropna(subset=['Rating'], how='all')
     train
```

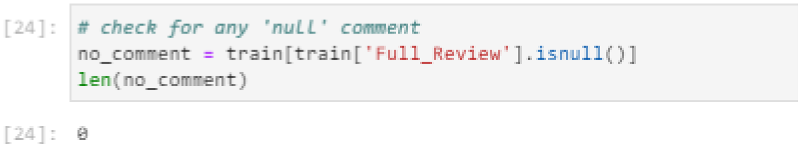
[9]:

	Short_Review	Rating	Full_Review
0	Fabulous!	5.0	i loved it, superb performance, awesome batter...
1	Super!	5.0	I wonder why nobody posted pics for snow white...
2	Just wow!	5.0	An amazing device especially for high & smooth...
10	Classy product	5.0	Nice product. Value for Money. Go for it if yo...
11	Fabulous!	5.0	Amazing product in price range , good sound qu...
...	...	...	...
21127	Classy product	5.0	1. Display quality is very good.\n2. Very thin...
21128	Good quality product	4.0	Nice watch but battery life is complete 4 days
21137	Mind-blowing purchase	4.0	Superb design and quality, appreciate before f...
21138	Waste of money!	4.0	Worst product delivered by flipkart . Not made...
21148	Terrific	4.0	Awesome Smart Watch\nAwesome Performance.\nA P...

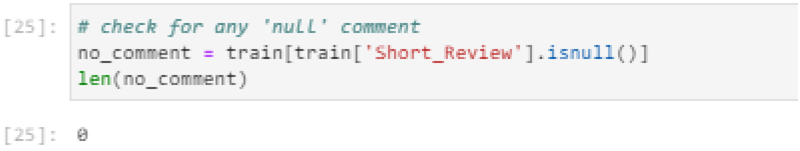
2663 rows × 3 columns

Dropping also the rows where we have NO RATINGS as we can't map the reviews with any RATING number.

After managing the missing values, we can see there is no missing value left in our data as shown below:



```
[24]: # check for any 'null' comment
      no_comment = train[train['Full_Review'].isnull()]
      len(no_comment)

[24]: 0
```



```
[25]: # check for any 'null' comment
      no_comment = train[train['Short_Review'].isnull()]
      len(no_comment)

[25]: 0
```

OK, no null values in any column. Great, we can move forward now!

We will then prepare the data for our ML learning, which means we need to preprocess and convert the short reviews and full reviews in kind of numerical data so that ML algorithm can be trained on it.

The steps we will follow are the following ones:

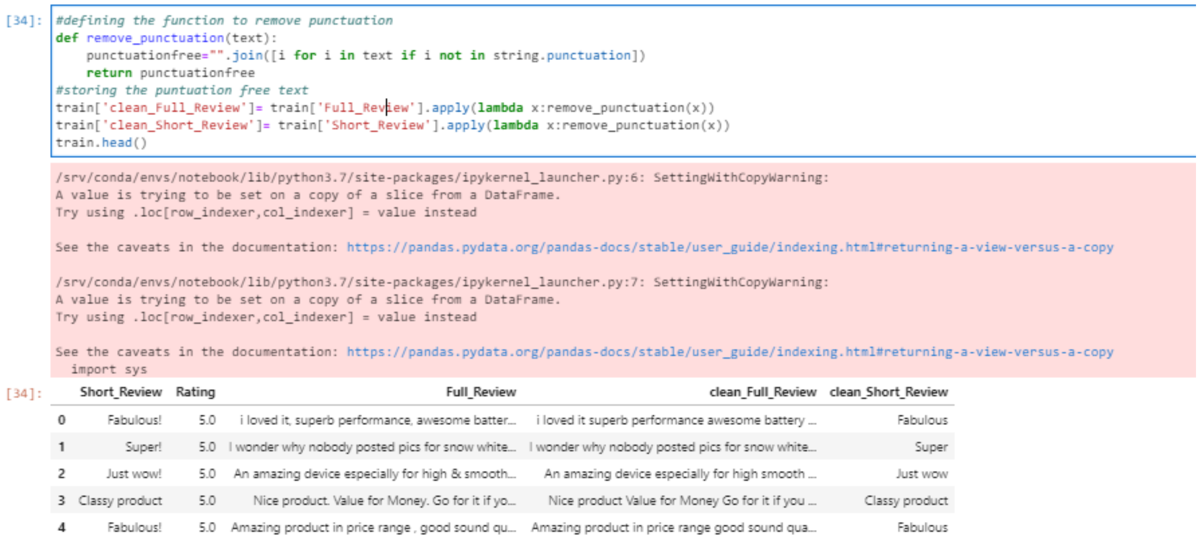
- remove_punctuation

-tokenization

- stopwords

- lemmatization

- word embedding(tf-idf)



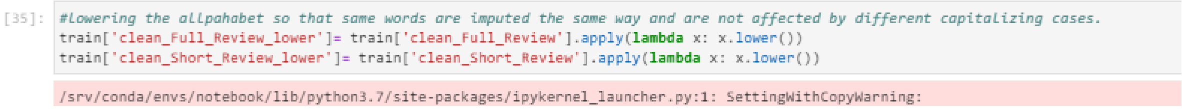
```
[34]: #defining the function to remove punctuation
      def remove_punctuation(text):
          punctuationfree="".join([i for i in text if i not in string.punctuation])
          return punctuationfree
      #storing the puntuation free text
      train['clean_Full_Review']= train['Full_Review'].apply(lambda x:remove_punctuation(x))
      train['clean_Short_Review']= train['Short_Review'].apply(lambda x:remove_punctuation(x))
      train.head()
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/srv/conda/envs/notebook/lib/python3.7/site-packages/ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

[34]:

	Short_Review	Rating	Full_Review	clean_Full_Review	clean_Short_Review
0	Fabulous!	5.0	i loved it, superb performance, awesome batter...	i loved it superb performance awesome battery ...	Fabulous
1	Super!	5.0	I wonder why nobody posted pics for snow white...	I wonder why nobody posted pics for snow white...	Super
2	Just wow!	5.0	An amazing device especially for high & smooth...	An amazing device especially for high smooth ...	Just wow
3	Classy product	5.0	Nice product. Value for Money. Go for it if yo...	Nice product Value for Money Go for it if you ...	Classy product
4	Fabulous!	5.0	Amazing product in price range , good sound qu...	Amazing product in price range good sound qua...	Fabulous

First, we remove the extra insignificant punctuation we have in our data and that will not help the algorithm in understand the pattern of the reviews data and for its training.

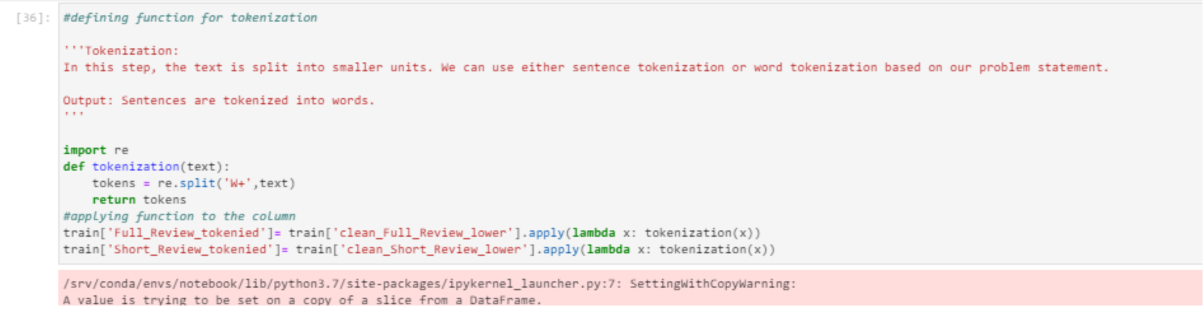


[35]:

```
#Lowering the allpahabet so that same words are imputed the same way and are not affected by different capitalizing cases.
train['clean_Full_Review_lower']= train['clean_Full_Review'].apply(lambda x: x.lower())
train['clean_Short_Review_lower']= train['clean_Short_Review'].apply(lambda x: x.lower())
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
```

Then, we will also lowercase our words so that same words are taken and annualized in the same way even if one alphabet of the word is in uppercase or not.

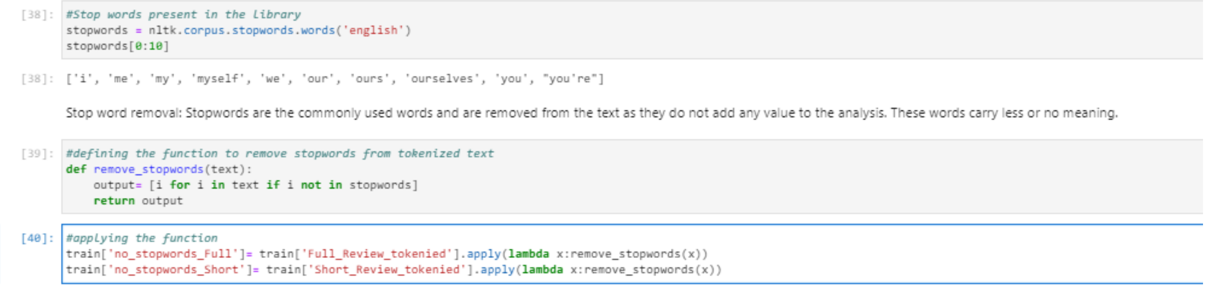


[36]:

```
#defining function for tokenization

'''Tokenization:
In this step, the text is split into smaller units. We can use either sentence tokenization or word tokenization based on our problem statement.

Output: Sentences are tokenized into words.
'''

import re
def tokenization(text):
    tokens = re.split('W+',text)
    return tokens
#applying function to the column
train['Full_Review_tokenied']= train['clean_Full_Review_lower'].apply(lambda x: tokenization(x))
train['Short_Review_tokenied']= train['clean_Short_Review_lower'].apply(lambda x: tokenization(x))
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

Then, we will tokenize our reviews. Which means, we will split our reviews into small units which will be words.



[38]:

```
#Stop words present in the Library
stopwords = nltk.corpus.stopwords.words('english')
stopwords[0:10]
```

[38]:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

Stop word removal: Stopwords are the commonly used words and are removed from the text as they do not add any value to the analysis. These words carry less or no meaning.

[39]:

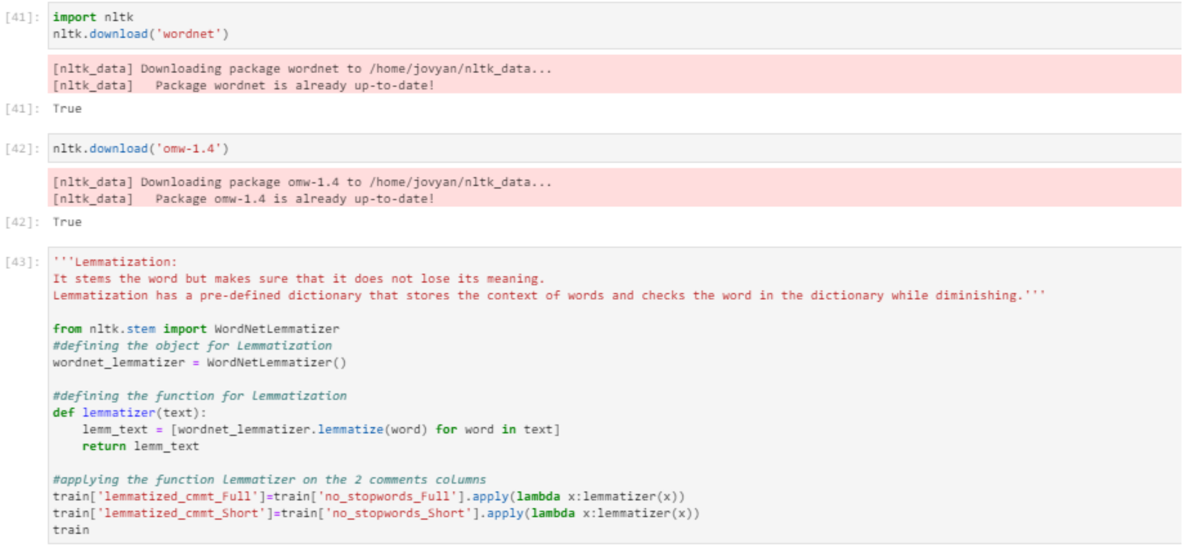
```
#defining the function to remove stopwords from tokenized text
def remove_stopwords(text):
    output= [i for i in text if i not in stopwords]
    return output
```

[40]:

```
#applying the function
train['no_stopwords_Full']= train['Full_Review_tokenied'].apply(lambda x:remove_stopwords(x))
train['no_stopwords_Short']= train['Short_Review_tokenied'].apply(lambda x:remove_stopwords(x))
```

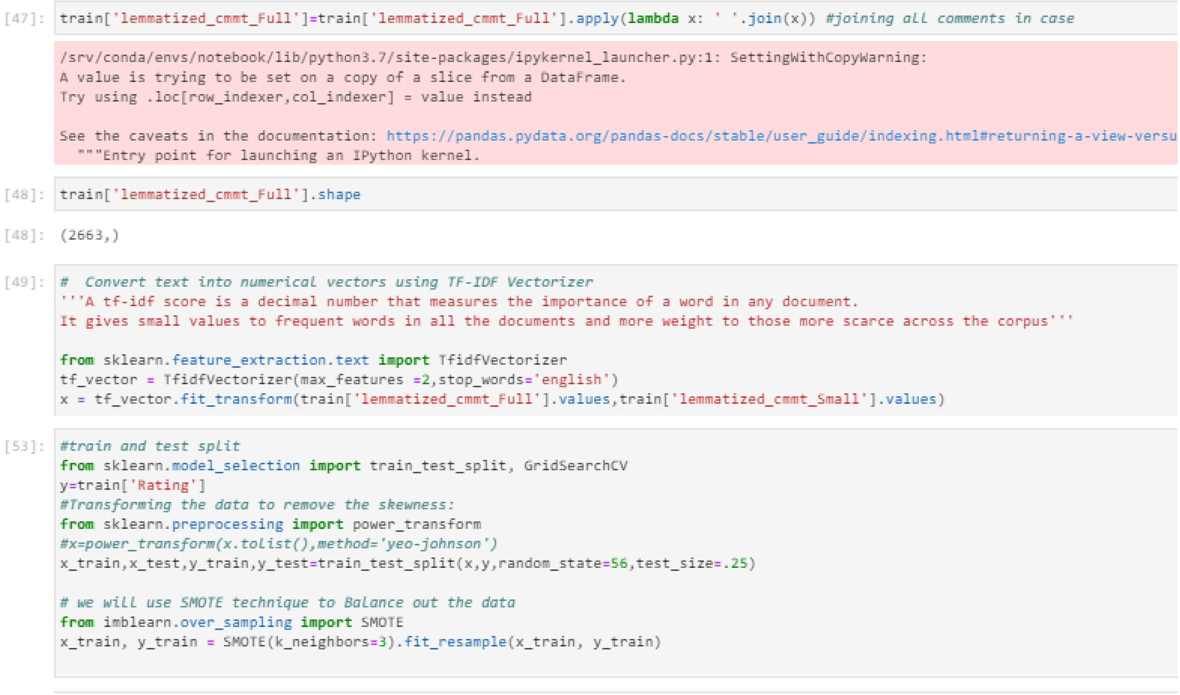
Then, we will apply Remove Stopwords function after having tokenized our sentences and separated in words.

Stopwords are the commonly used words and are removed from the text as they do not add any value to the analysis. These words carry less or no meaning. Hence, we will remove the stopwords found in the reviews.



```
[41]: import nltk
      nltk.download('wordnet')

      [nltk_data] Downloading package wordnet to /home/jovyan/nltk_data...
      [nltk_data]   Package wordnet is already up-to-date!

[41]: True

[42]: nltk.download('omw-1.4')

      [nltk_data] Downloading package omw-1.4 to /home/jovyan/nltk_data...
      [nltk_data]   Package omw-1.4 is already up-to-date!

[42]: True

[43]: '''Lemmatization:
      It stems the word but makes sure that it does not lose its meaning.
      Lemmatization has a pre-defined dictionary that stores the context of words and checks the word in the dictionary while diminishing.'''

      from nltk.stem import WordNetLemmatizer
      #defining the object for Lemmatization
      wordnet_lemmatizer = WordNetLemmatizer()

      #defining the function for Lemmatization
      def lemmatizer(text):
          lemm_text = [wordnet_lemmatizer.lemmatize(word) for word in text]
          return lemm_text

      #applying the function Lemmatizer on the 2 comments columns
      train['lemmatized_cmmt_Full']=train['no_stopwords_Full'].apply(lambda x:lemmatizer(x))
      train['lemmatized_cmmt_Short']=train['no_stopwords_Short'].apply(lambda x:lemmatizer(x))
      train
```

After having removed stopwords, we will use WordNetLemmatizer so that we can stems the word but makes sure that it does not lose its meaning. Lemmatization has a pre-defined dictionary that stores the context of words and checks the word in the dictionary while diminishing.

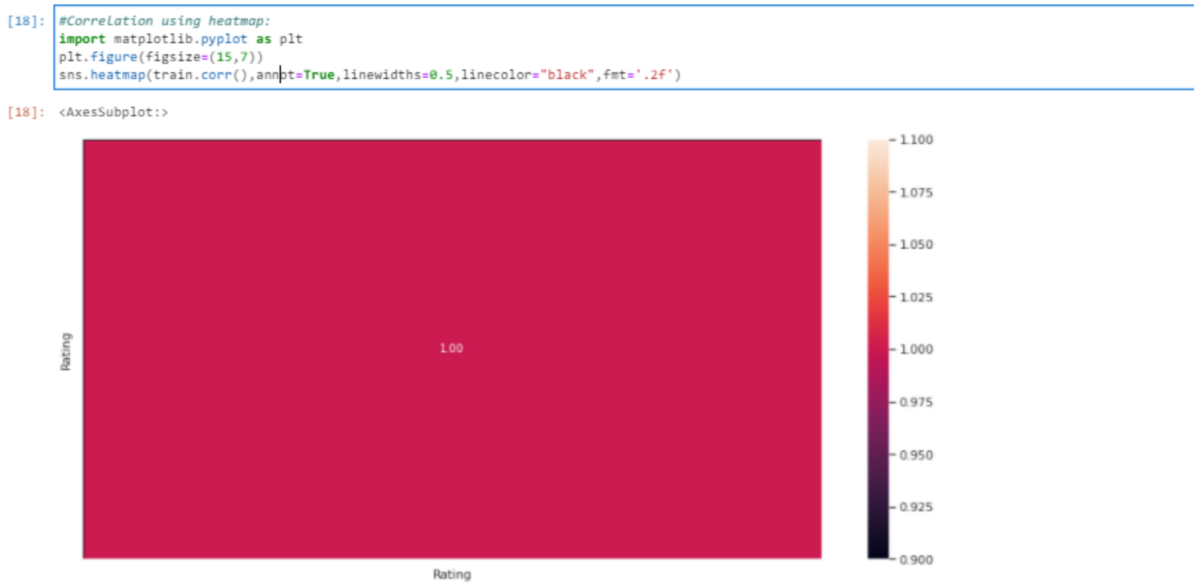


```
[47]: train['lemmatized_cmmt_Full']=train['lemmatized_cmmt_Full'].apply(lambda x: ' '.join(x)) #joining all comments in case

      /srv/conda/envs/notebook/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
      A value is trying to be set on a copy of a slice from a DataFrame.
      Try using .loc[row_indexer,col_indexer] = value instead

      See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
        """Entry point for launching an IPython kernel.

[48]: train['lemmatized_cmmt_Full'].shape

[48]: (2663,)

[49]: #  Convert text into numerical vectors using TF-IDF Vectorizer
      '''A tf-idf score is a decimal number that measures the importance of a word in any document.
      It gives small values to frequent words in all the documents and more weight to those more scarce across the corpus'''

      from sklearn.feature_extraction.text import TfidfVectorizer
      tf_vector = TfidfVectorizer(max_features =2,stop_words='english')
      x = tf_vector.fit_transform(train['lemmatized_cmmt_Full'].values,train['lemmatized_cmmt_Small'].values)

[53]: #train and test split
      from sklearn.model_selection import train_test_split, GridSearchCV
      y=train['Rating']
      #Transforming the data to remove the skewness:
      from sklearn.preprocessing import power_transform
      #x=power_transform(x.tolist(),method='yeo-johnson')
      x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=56,test_size=.25)

      # we will use SMOTE technique to Balance out the data
      from imblearn.over_sampling import SMOTE
      x_train, y_train = SMOTE(k_neighbors=3).fit_resample(x_train, y_train)
```

And finally, before our ML training we need to convert text into numerical vectors using TF-IDF Vectorizer. A tf-idf score is a decimal number that measures the importance of a word in any document. It gives small values to frequent words in all the documents and more weight to those scarcer across the corpus of text.

- Data Inputs- Logic- Output Relationships

Describe the relationship behind the data input, its format, the logic in between and the output. Describe how the input affects the output.



```
[18]: #Correlation using heatmap:
      import matplotlib.pyplot as plt
      plt.figure(figsize=(15,7))
      sns.heatmap(train.corr(),annot=True,linewidths=0.5,linecolor="black",fmt='.2f')

[18]: <AxesSubplot:>
```

On one hand, No correlation is expected between our numerical features as we only have one numerical feature. No other numerical feature to correlate with.

On the other hand, we do expect some relation between the rating and the Short and Full Review. There must be a correlation. For that, first we need to pre-process the reviews through NLTK library and convert the reviews/strings into numerical data so that ML algorithm can be trained on it. The pre-process steps are:

- remove_punctuation

-tokenization

- stopwords

- lemmatization

- word embedding(tf-idf)

Which are already explained and discussed in the previous section “Pre-processing Done”.

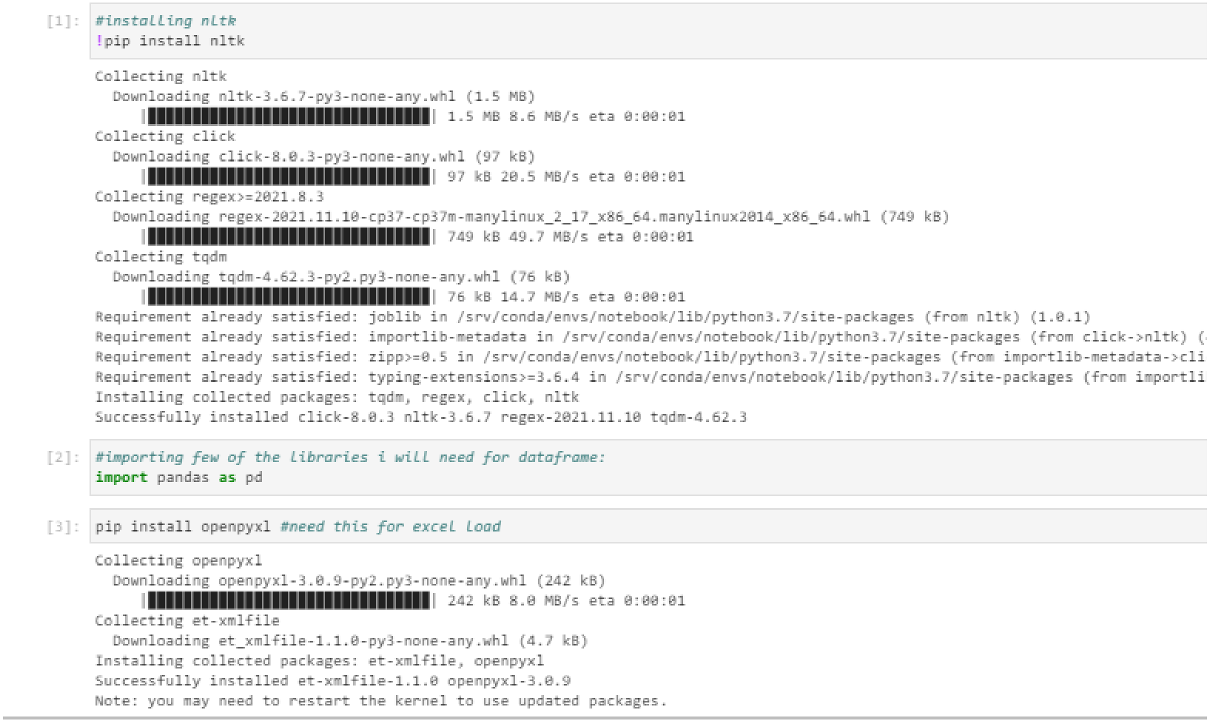
- ## State the set of assumptions (if any) related to the problem under consideration

The assumption taken by me were:

Not exactly an assumption, but yes consideration: More stopwords we remove, less comment length will go to the training algorithm which is good.

- ## Hardware and Software Requirements and Tools Used

Listing down the hardware and software requirements along with the tools, libraries and packages used. Describe all the software tools used along with a detailed description of tasks done with those tools.

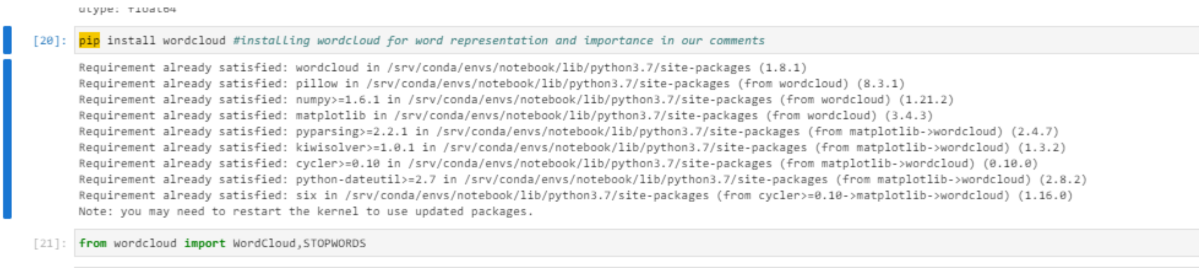


```
[1]: #installing nltk
     !pip install nltk

     Collecting nltk
       Downloading nltk-3.6.7-py3-none-any.whl (1.5 MB)
          |████████████████████████████████| 1.5 MB 8.6 MB/s eta 0:00:01
     Collecting click
       Downloading click-8.0.3-py3-none-any.whl (97 kB)
          |████████████████████████████████| 97 kB 20.5 MB/s eta 0:00:01
     Collecting regex>=2021.8.3
       Downloading regex-2021.11.10-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (749 kB)
          |████████████████████████████████| 749 kB 49.7 MB/s eta 0:00:01
     Collecting tqdm
       Downloading tqdm-4.62.3-py2.py3-none-any.whl (76 kB)
          |████████████████████████████████| 76 kB 14.7 MB/s eta 0:00:01
     Requirement already satisfied: joblib in /srv/conda/envs/notebook/lib/python3.7/site-packages (from nltk) (1.0.1)
     Requirement already satisfied: importlib-metadata in /srv/conda/envs/notebook/lib/python3.7/site-packages (from click->nltk) (
     Requirement already satisfied: zipp>=0.5 in /srv/conda/envs/notebook/lib/python3.7/site-packages (from importlib-metadata->cli
     Requirement already satisfied: typing-extensions>=3.6.4 in /srv/conda/envs/notebook/lib/python3.7/site-packages (from importli
     Installing collected packages: tqdm, regex, click, nltk
     Successfully installed click-8.0.3 nltk-3.6.7 regex-2021.11.10 tqdm-4.62.3

[2]: #importing few of the libraries i will need for dataframe:
     import pandas as pd

[3]: pip install openpyxl #need this for excel load

     Collecting openpyxl
       Downloading openpyxl-3.0.9-py2.py3-none-any.whl (242 kB)
          |████████████████████████████████| 242 kB 8.0 MB/s eta 0:00:01
     Collecting et-xmlfile
       Downloading et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)
     Installing collected packages: et-xmlfile, openpyxl
     Successfully installed et-xmlfile-1.1.0 openpyxl-3.0.9
     Note: you may need to restart the kernel to use updated packages.
```

Installed nltk library for natural language processing.

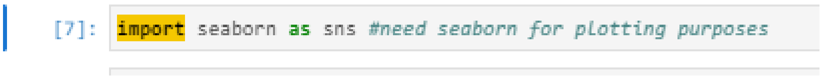
Loaded pandas as it is a software library written for the Python programming language for data manipulation and analysis.

Installed openpyxl for excel data load.



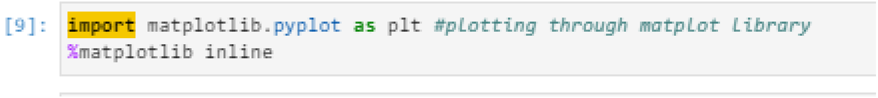
```
dtype: float64
[20]: pip install wordcloud #installing wordcloud for word representation and importance in our comments
      Requirement already satisfied: wordcloud in /srv/conda/envs/notebook/lib/python3.7/site-packages (1.8.1)
      Requirement already satisfied: pillow in /srv/conda/envs/notebook/lib/python3.7/site-packages (from wordcloud) (8.3.1)
      Requirement already satisfied: numpy>=1.6.1 in /srv/conda/envs/notebook/lib/python3.7/site-packages (from wordcloud) (1.21.2)
      Requirement already satisfied: matplotlib in /srv/conda/envs/notebook/lib/python3.7/site-packages (from wordcloud) (3.4.3)
      Requirement already satisfied: pyparsing>=2.2.1 in /srv/conda/envs/notebook/lib/python3.7/site-packages (from matplotlib->wordcloud) (2.4.7)
      Requirement already satisfied: kiwisolver>=1.0.1 in /srv/conda/envs/notebook/lib/python3.7/site-packages (from matplotlib->wordcloud) (1.3.2)
      Requirement already satisfied: cycler>=0.10 in /srv/conda/envs/notebook/lib/python3.7/site-packages (from matplotlib->wordcloud) (0.10.0)
      Requirement already satisfied: python-dateutil>=2.7 in /srv/conda/envs/notebook/lib/python3.7/site-packages (from matplotlib->wordcloud) (2.8.2)
      Requirement already satisfied: six in /srv/conda/envs/notebook/lib/python3.7/site-packages (from cycler>=0.10->matplotlib->wordcloud) (1.16.0)
      Note: you may need to restart the kernel to use updated packages.
[21]: from wordcloud import WordCloud,STOPWORDS
```

Loaded world cloud library to show the words that appears most in our natural language text.



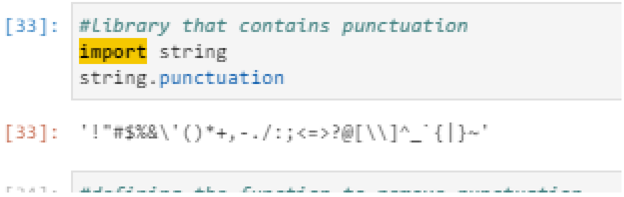
```
[7]: import seaborn as sns #need seaborn for plotting purposes
```

Imported seaborn for plotting purposes



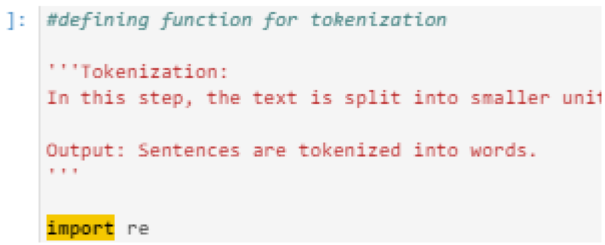
```
[9]: import matplotlib.pyplot as plt #plotting through matplot library
     %matplotlib inline
```

Loaded also matplotlib to plot the data and executed %inline so that the plot is visible when executing the code.



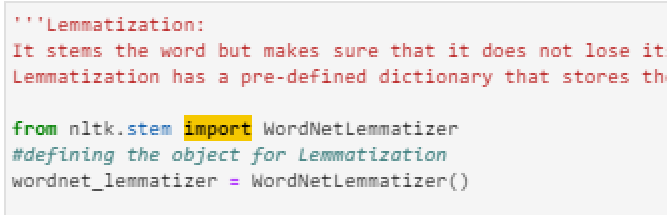
```
[33]: #library that contains punctuation
      import string
      string.punctuation
[33]: '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
[34]: #defining the function to remove punctuation
```

Loaded library that contains punctuation.



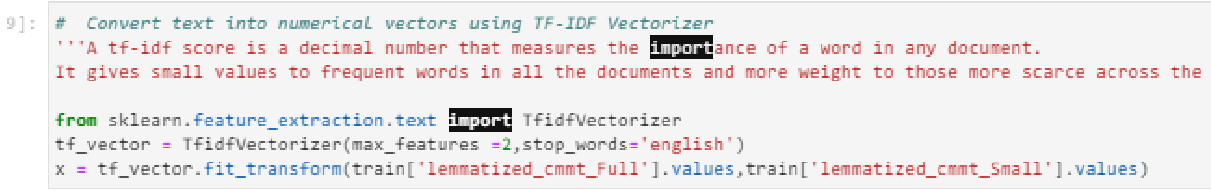
```
]: #defining function for tokenization

   '''Tokenization:
   In this step, the text is split into smaller unit

   Output: Sentences are tokenized into words.
   '''

   import re
```

Imported re for regular expressions.



```
'''Lemmatization:
It stems the word but makes sure that it does not lose it
Lemmatization has a pre-defined dictionary that stores th

from nltk.stem import WordNetLemmatizer
#defining the object for Lemmatization
wordnet_lemmatizer = WordNetLemmatizer()
```

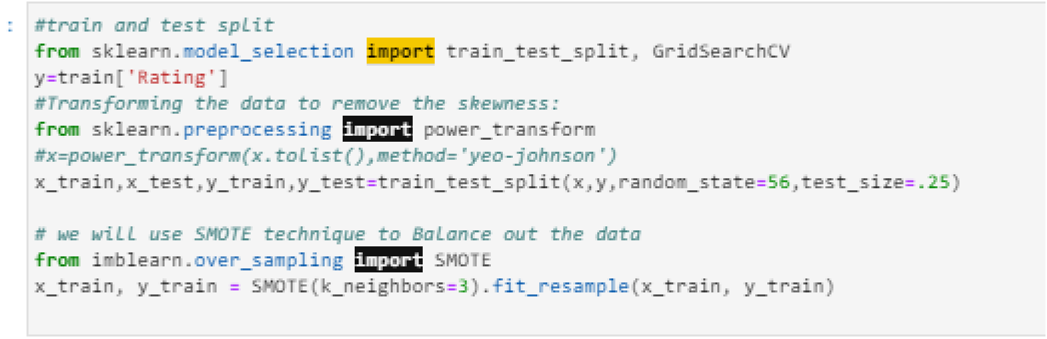
Imported WordNetLemmatizer for Lemmatization which has a pre-defined dictionary that stores the context of words and checks the word in the dictionary while diminishing. It stems the word but makes sure that it does not lose its meaning.



```
9]: #  Convert text into numerical vectors using TF-IDF Vectorizer
    '''A tf-idf score is a decimal number that measures the importance of a word in any document.
    It gives small values to frequent words in all the documents and more weight to those more scarce across the

    from sklearn.feature_extraction.text import TfidfVectorizer
    tf_vector = TfidfVectorizer(max_features =2,stop_words='english')
    x = tf_vector.fit_transform(train['lemmatized_cmmt_Full'].values,train['lemmatized_cmmt_Small'].values)
```

Importing TfidfVectorizer for vectorization purposes. A tf-idf score is a decimal number that measures the importance of a word in any document.

It gives small values to frequent words in all the documents and more weight to those more scarce across the corpus.

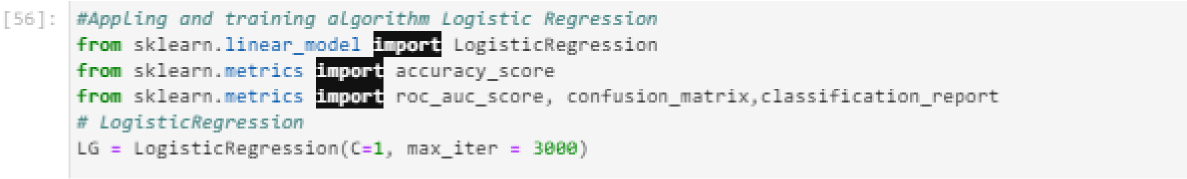


```
: #train and test split
  from sklearn.model_selection import train_test_split, GridSearchCV
  y=train['Rating']
  #Transforming the data to remove the skewness:
  from sklearn.preprocessing import power_transform
  #x=power_transform(x.tolist(),method='yeo-johnson')
  x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=56,test_size=.25)

  # we will use SMOTE technique to Balance out the data
  from imblearn.over_sampling import SMOTE
  x_train, y_train = SMOTE(k_neighbors=3).fit_resample(x_train, y_train)
```

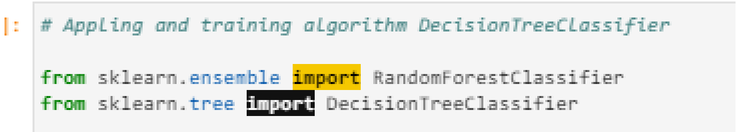
Imported train_test_split for training and testing split of the data and GridSeachCV for best parameter findings. We can define it as exhaustive search over specified parameter values for an estimator.

Imported power_transform for skewness adjustment.

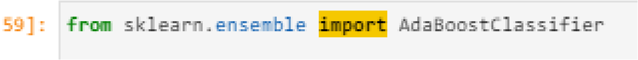
Imported SMOTE for imbalance adjustment.



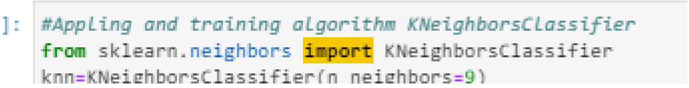
```
[56]: #Appling and training algorithm Logistic Regression
      from sklearn.linear_model import LogisticRegression
      from sklearn.metrics import accuracy_score
      from sklearn.metrics import roc_auc_score, confusion_matrix,classification_report
      # LogisticRegression
      LG = LogisticRegression(C=1, max_iter = 3000)
```



```
]: # Appling and training algorithm DecisionTreeClassifier

   from sklearn.ensemble import RandomForestClassifier
   from sklearn.tree import DecisionTreeClassifier
```



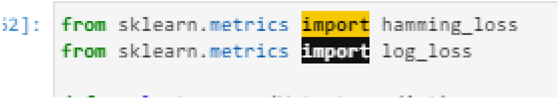
```
59]: from sklearn.ensemble import AdaBoostClassifier
```



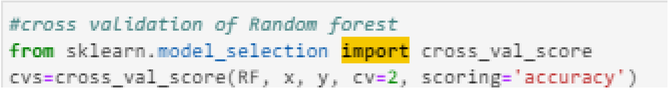
```
]: #Appling and training algorithm KNeighborsClassifier
   from sklearn.neighbors import KNeighborsClassifier
   knn=KNeighborsClassifier(n_neighbors=9)
```

Imported several algorithms such as LogisticRegression, RandomForestClassifier, DecisiontreeClassifier, AdaboostClassifier, KNeighborsClassifier for training and testing purposes.

Imported roc_auc_score, confusion_matrix,classification_report for results analysis and evaluation.



```
52]: from sklearn.metrics import hamming_loss
     from sklearn.metrics import log_loss
```



```
#cross validation of Random forest
from sklearn.model_selection import cross_val_score
cvs=cross_val_score(RF, x, y, cv=2, scoring='accuracy')
```

For results evaluation and comparison, loaded several functions from sklearn.metrics such as accuracy_score, f1_score, recall_score, precision_score, roc_auc_score, roc_curve, confusion_matrix, cross_val_score, hamming_loss.

```
8]: #Library for saving data
    import pickle
    filename='RF.pickle'
    pickle.dump(RF,open(filename,'wb'))
```

And finally loaded pickle for algorithm saving purposes.

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

Basically, we tried to clean the reviews as much as possible, understand the relations and correlation between each feature and predict the rating of a review through several classification algorithms.

Compared the means and statistical deviation in order to get some views on the distribution and also did some plotting to see visually the distribution of the target data which is Rating.

And mostly importantly, we pre-processed and prepared the data and reviews for our machine learning through different techniques of nltk.

Approaches like cross validation and classification reports are done to analyse the results of algorithm and training performance.

We used accuracy, precision, recall, f1 score, hamming loss and log loss for validation and results evaluation and comparison.

- ## Testing of Identified Approaches (Algorithms)

  Listing down all the algorithms used for the training and testing:

  I used and compared the following algorithms:

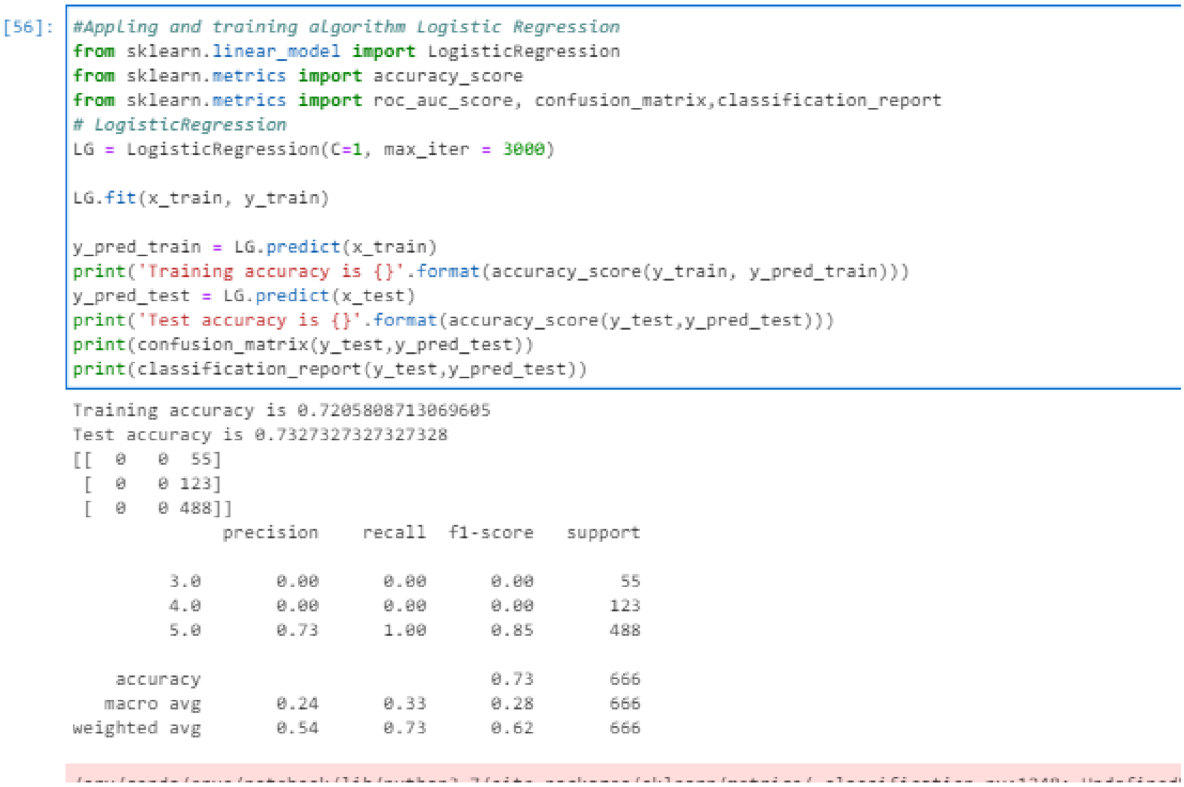
  - Logistic Regression
  - Decision Tree Classifier
  - Random Forest Classifier
  - AdaBoost Classifier
  - KNeighbor Classifier

  .

- ## Run and Evaluate selected models

  Describe all the algorithms used along with the snapshot of their code and what were the results observed over different evaluation metrics.

  - As we mentioned we used Logistic Regression which is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.



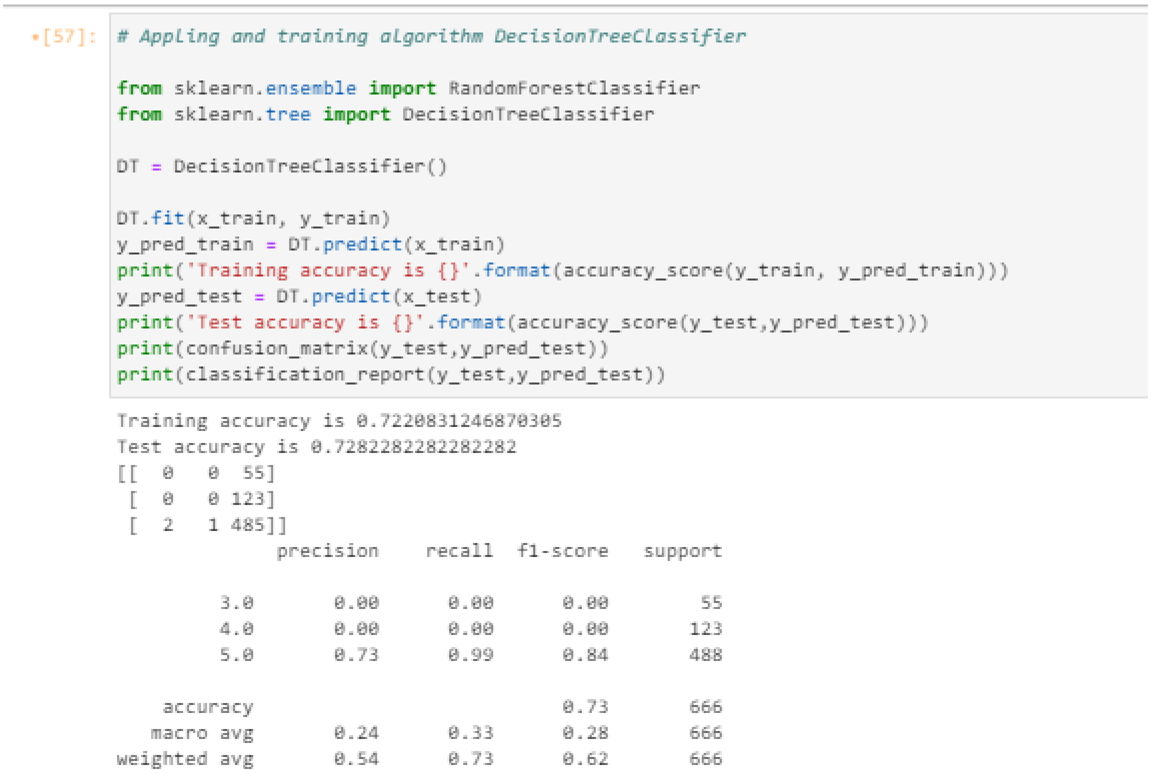
```
[56]: #Appling and training algorithm Logistic Regression
      from sklearn.linear_model import LogisticRegression
      from sklearn.metrics import accuracy_score
      from sklearn.metrics import roc_auc_score, confusion_matrix,classification_report
      # LogisticRegression
      LG = LogisticRegression(C=1, max_iter = 3000)

      LG.fit(x_train, y_train)

      y_pred_train = LG.predict(x_train)
      print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
      y_pred_test = LG.predict(x_test)
      print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
      print(confusion_matrix(y_test,y_pred_test))
      print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.7205808713069605
Test accuracy is 0.7327327327327328
[[  0   0  55]
 [  0   0 123]
 [  0   0 488]]
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00        55
         4.0       0.00      0.00      0.00       123
         5.0       0.73      1.00      0.85       488

    accuracy                           0.73       666
   macro avg       0.24      0.33      0.28       666
weighted avg       0.54      0.73      0.62       666

/opt/conda/envs/notebook/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1248: Undefined
```

- Decision tree works by successively splitting the dataset into small segments until the target variable are the same or until the dataset can no longer be split. It's a greedy algorithm which make the best decision at the given time without concern for the global optimality

The construction is similar:

1. Assign all training instances to the root of the tree. Set current node to root node.
2. Find the split feature and split value based on the split criterion such as information gain, information gain ratio or gini coefficient.
3. Partition all data instances at the node based on the split feature and threshold value.
4. Denote each partition as a child node of the current node.
5. For each child node:
    1. If the child node is "pure" (has instances from only one class), tag it as a leaf and return.
    2. Else, set the child node as the current node and recurse to step 2.

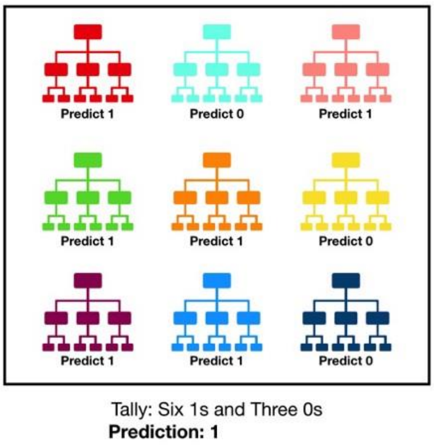


```
•[57]: # Appling and training algorithm DecisionTreeClassifier

       from sklearn.ensemble import RandomForestClassifier
       from sklearn.tree import DecisionTreeClassifier

       DT = DecisionTreeClassifier()

       DT.fit(x_train, y_train)
       y_pred_train = DT.predict(x_train)
       print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
       y_pred_test = DT.predict(x_test)
       print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
       print(confusion_matrix(y_test,y_pred_test))
       print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.7220831246870305
Test accuracy is 0.7282282282282282
[[  0   0  55]
 [  0   0 123]
 [  2   1 485]]
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00        55
         4.0       0.00      0.00      0.00       123
         5.0       0.73      0.99      0.84       488

    accuracy                           0.73       666
   macro avg       0.24      0.33      0.28       666
weighted avg       0.54      0.73      0.62       666
```

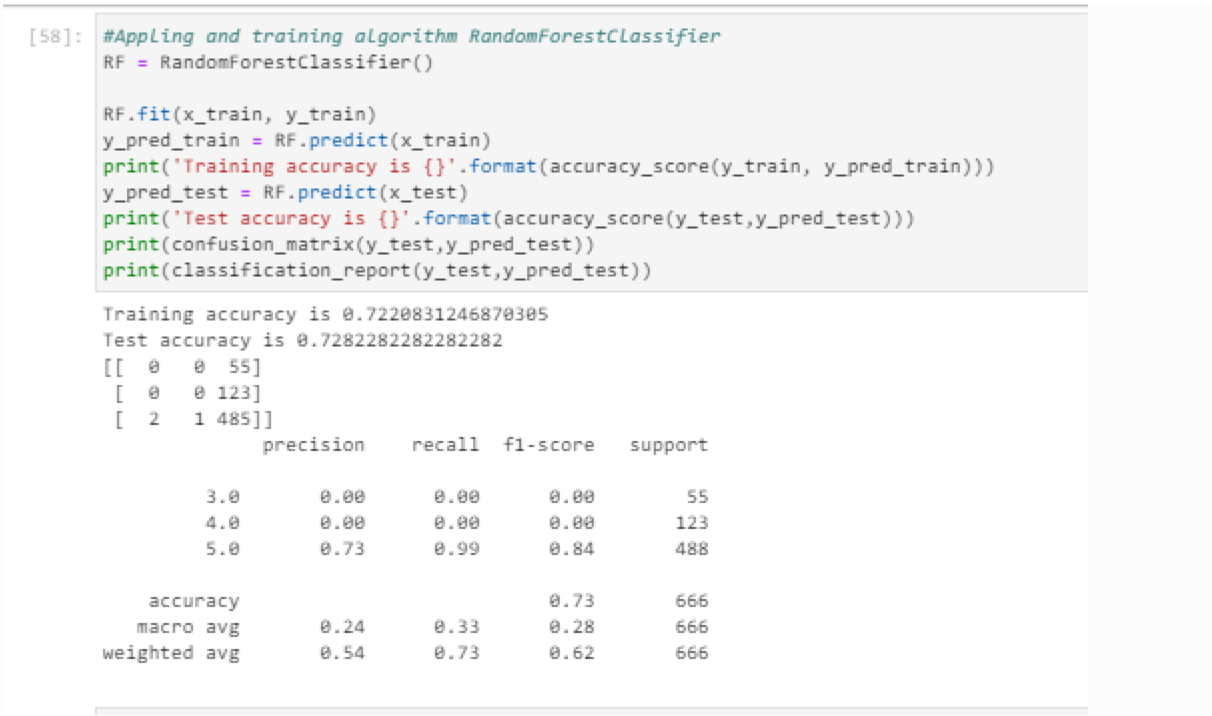
Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model’s prediction (see figure below).



Visualization of a Random Forest Model Making a Prediction

The fundamental concept behind random forest is a simple: wisdom of crowds. The reason is that a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The low correlation between models is the key. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all error don't occur in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

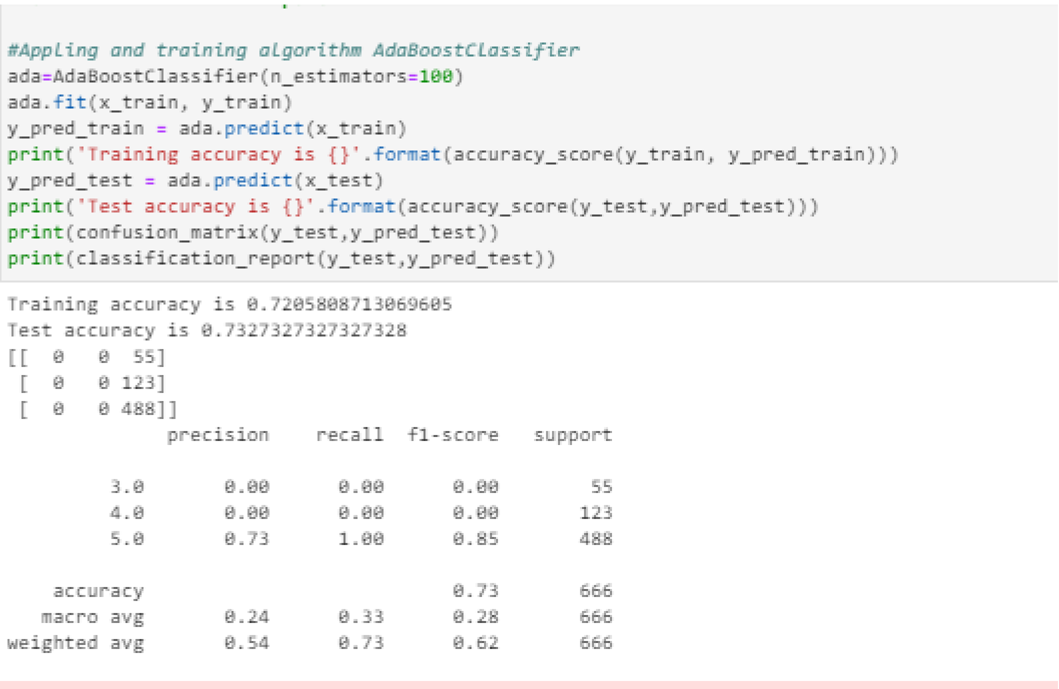


```
[58]: #Appling and training algorithm RandomForestClassifier
      RF = RandomForestClassifier()

      RF.fit(x_train, y_train)
      y_pred_train = RF.predict(x_train)
      print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
      y_pred_test = RF.predict(x_test)
      print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
      print(confusion_matrix(y_test,y_pred_test))
      print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.7220831246870305
Test accuracy is 0.7282282282282282
[[  0   0  55]
 [  0   0 123]
 [  2   1 485]]
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00        55
         4.0       0.00      0.00      0.00       123
         5.0       0.73      0.99      0.84       488

    accuracy                           0.73       666
   macro avg       0.24      0.33      0.28       666
weighted avg       0.54      0.73      0.62       666
```

An AdaBoost Classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

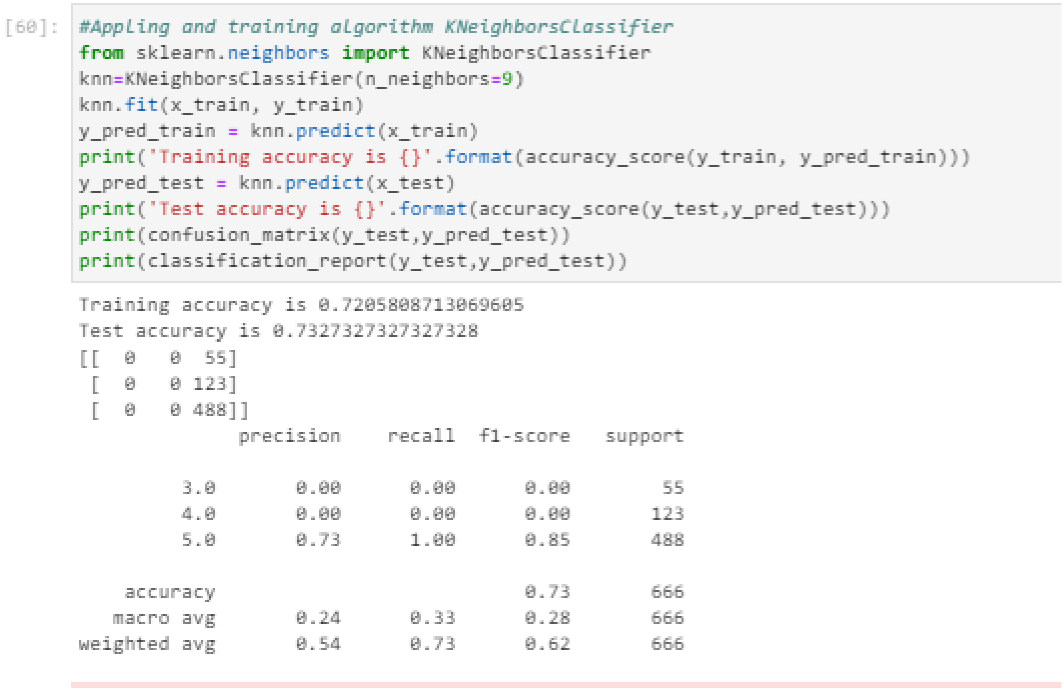
AdaBoost algorithm, called Adaptive Boosting, as the weights are re-assigned to each instance, with higher weights assigned to incorrectly classified instances. Boosting is used to reduce bias as well as variance for supervised learning. It works on the principle of learners growing sequentially. Except for the first, each subsequent learner is grown from previously grown learners. In simple words, weak learners are converted into strong ones.



```
#Appling and training algorithm AdaBoostClassifier
ada=AdaBoostClassifier(n_estimators=100)
ada.fit(x_train, y_train)
y_pred_train = ada.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = ada.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.7205808713069605
Test accuracy is 0.7327327327327328
[[  0   0  55]
 [  0   0 123]
 [  0   0 488]]
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00        55
         4.0       0.00      0.00      0.00       123
         5.0       0.73      1.00      0.85       488

    accuracy                           0.73       666
   macro avg       0.24      0.33      0.28       666
weighted avg       0.54      0.73      0.62       666
```

K-Nearest-Neighbor is a Machine Learning supervised type instance-based algorithm. It can be used to classify new samples (discrete values) or to predict (regression, continuous values). Being a simple method, it is ideal to enter the world of Machine Learning. It essentially serves to classify values by looking for the "most similar" data points (by proximity) learned in the training stage and making guesses for new points based on that classification.



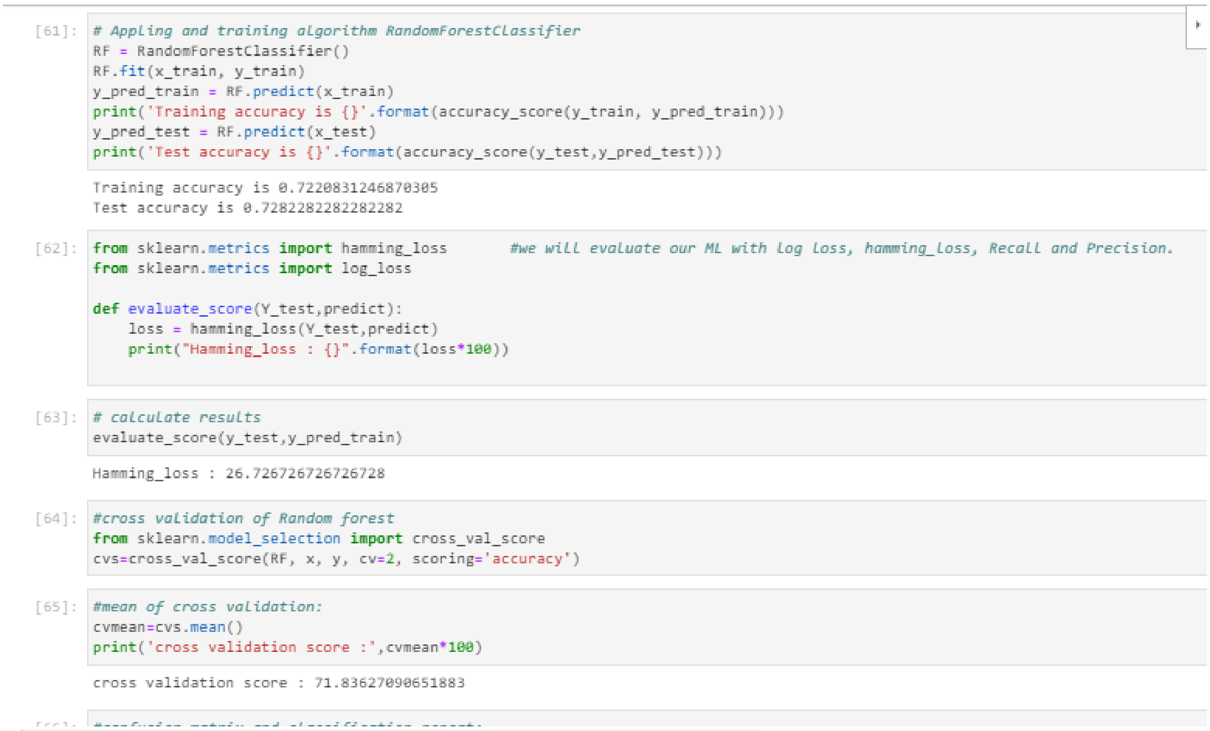
```
[60]: #Appling and training algorithm KNeighborsClassifier
      from sklearn.neighbors import KNeighborsClassifier
      knn=KNeighborsClassifier(n_neighbors=9)
      knn.fit(x_train, y_train)
      y_pred_train = knn.predict(x_train)
      print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
      y_pred_test = knn.predict(x_test)
      print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
      print(confusion_matrix(y_test,y_pred_test))
      print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.7205808713069605
Test accuracy is 0.7327327327327328
[[  0   0  55]
 [  0   0 123]
 [  0   0 488]]
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00        55
         4.0       0.00      0.00      0.00       123
         5.0       0.73      1.00      0.85       488

    accuracy                           0.73       666
   macro avg       0.24      0.33      0.28       666
weighted avg       0.54      0.73      0.62       666
```

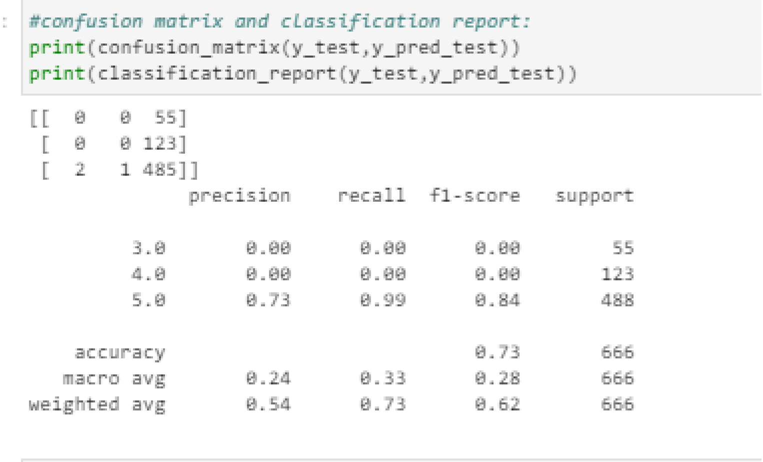
After analysing and comparing the results, we decided to go with Random Forest algorithm as it gave us the best results such as accuracy, precision,recall and therefore f1Score.

- ## Key Metrics for success in solving problem under consideration

The key metrics I used in order to compare the results were Training and Testing accuracy, Hamming Loss and Logg Loss. I also used and compared cross validation score alongside the precision, the recall score and the f1 score as shown in below example:



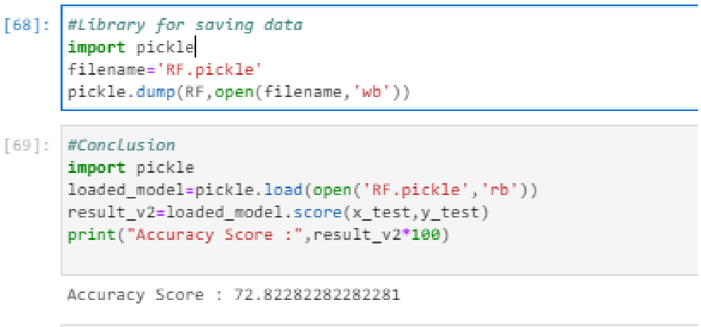
```
[61]: # Appling and training algorithm RandomForestClassifier
      RF = RandomForestClassifier()
      RF.fit(x_train, y_train)
      y_pred_train = RF.predict(x_train)
      print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
      y_pred_test = RF.predict(x_test)
      print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))

      Training accuracy is 0.7220831246870305
      Test accuracy is 0.7282282282282282

[62]: from sklearn.metrics import hamming_loss      #we will evaluate our ML with log loss, hamming_loss, Recall and Precision.
      from sklearn.metrics import log_loss

      def evaluate_score(Y_test,predict):
          loss = hamming_loss(Y_test,predict)
          print("Hamming_loss : {}".format(loss*100))

[63]: # calculate results
      evaluate_score(y_test,y_pred_train)

      Hamming_loss : 26.726726726726728

[64]: #cross validation of Random forest
      from sklearn.model_selection import cross_val_score
      cvs=cross_val_score(RF, x, y, cv=2, scoring='accuracy')

[65]: #mean of cross validation:
      cvmean=cvs.mean()
      print('cross validation score :',cvmean*100)

      cross validation score : 71.83627090651883
```



```
: #confusion matrix and classification report:
  print(confusion_matrix(y_test,y_pred_test))
  print(classification_report(y_test,y_pred_test))

  [[  0   0  55]
   [  0   0 123]
   [  2   1 485]]
                precision    recall  f1-score   support

           3.0       0.00      0.00      0.00        55
           4.0       0.00      0.00      0.00       123
           5.0       0.73      0.99      0.84       488

      accuracy                           0.73       666
     macro avg       0.24      0.33      0.28       666
  weighted avg       0.54      0.73      0.62       666
```

Once checked the weights of the words of a comment, we fitted the tf_vector on the both feature including reviews and checked and predicted on test data.

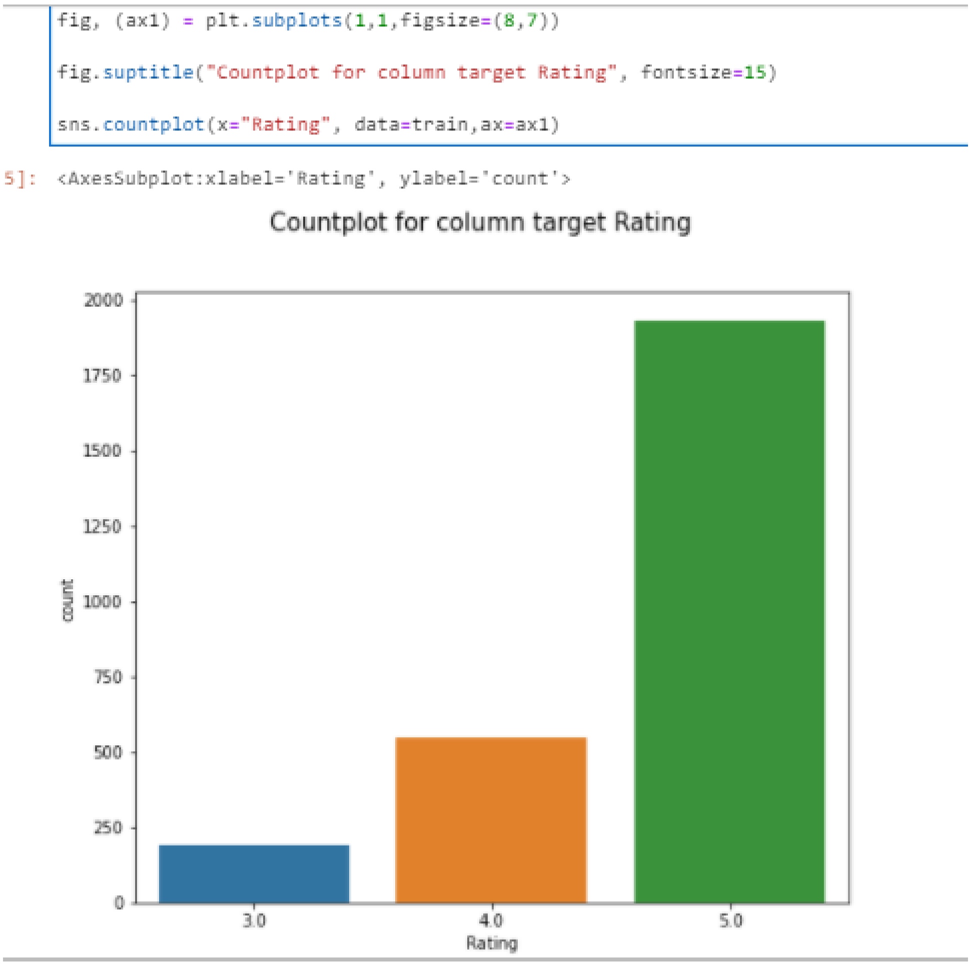
Then finally, we saved the model as “RF.pickle” so that the technical engineer can implement wherever is needed.



```
[68]: #Library for saving data
      import pickle
      filename='RF.pickle'
      pickle.dump(RF,open(filename,'wb'))

[69]: #Conclusion
      import pickle
      loaded_model=pickle.load(open('RF.pickle','rb'))
      result_v2=loaded_model.score(x_test,y_test)
      print("Accuracy Score :",result_v2*100)

      Accuracy Score : 72.82282282282281
```

- Visualizations

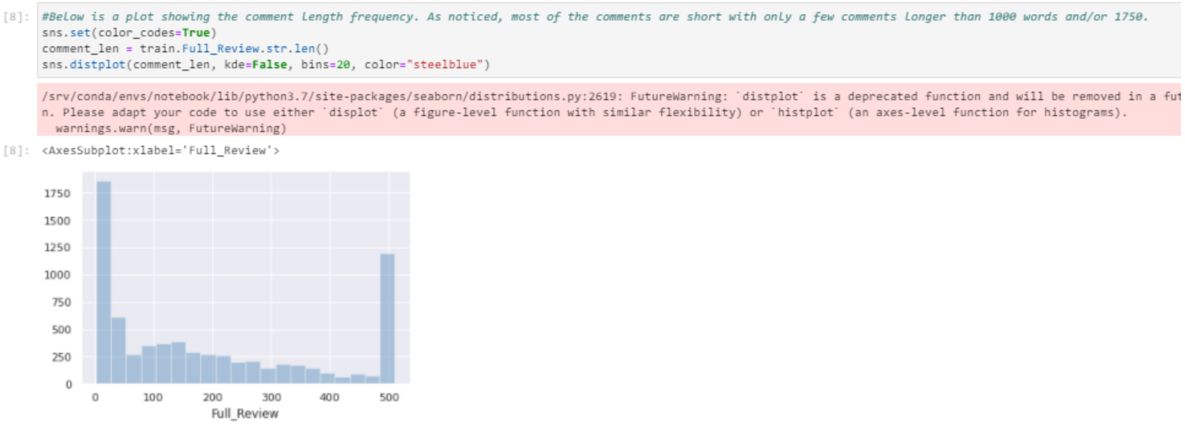
Mention all the plots made along with their pictures and what were the inferences and observations obtained from those. Describe them in detail.



```
fig, (ax1) = plt.subplots(1,1,figsize=(8,7))

fig.suptitle("Countplot for column target Rating", fontsize=15)

sns.countplot(x="Rating", data=train,ax=ax1)
```

```
5]: <AxesSubplot:xlabel='Rating', ylabel='count'>
```

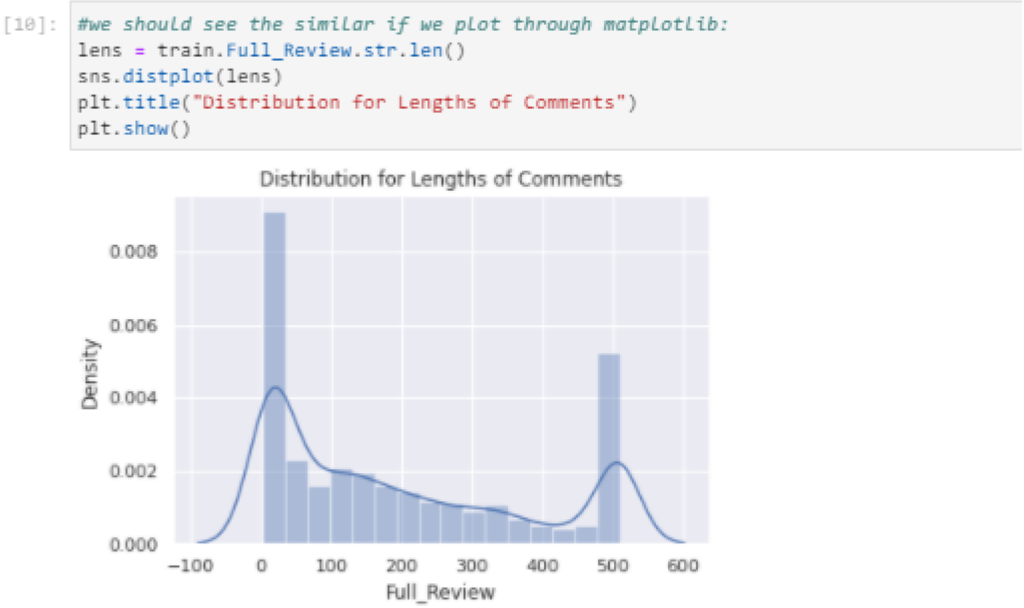
We have an imbalance in our target feature, so we will re-balance it through the tecnique SMOTE



```
[8]: #Below is a plot showing the comment length frequency. As noticed, most of the comments are short with only a few comments longer than 1000 words and/or 1750.
sns.set(color_codes=True)
comment_len = train.Full_Review.str.len()
sns.distplot(comment_len, kde=False, bins=20, color="steelblue")
```

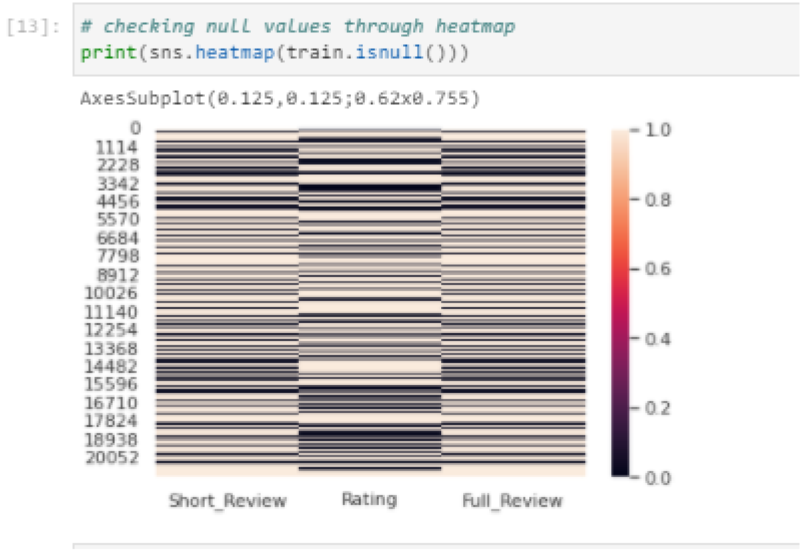
```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a fut
n. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
[8]: <AxesSubplot:xlabel='Full_Review'>
```



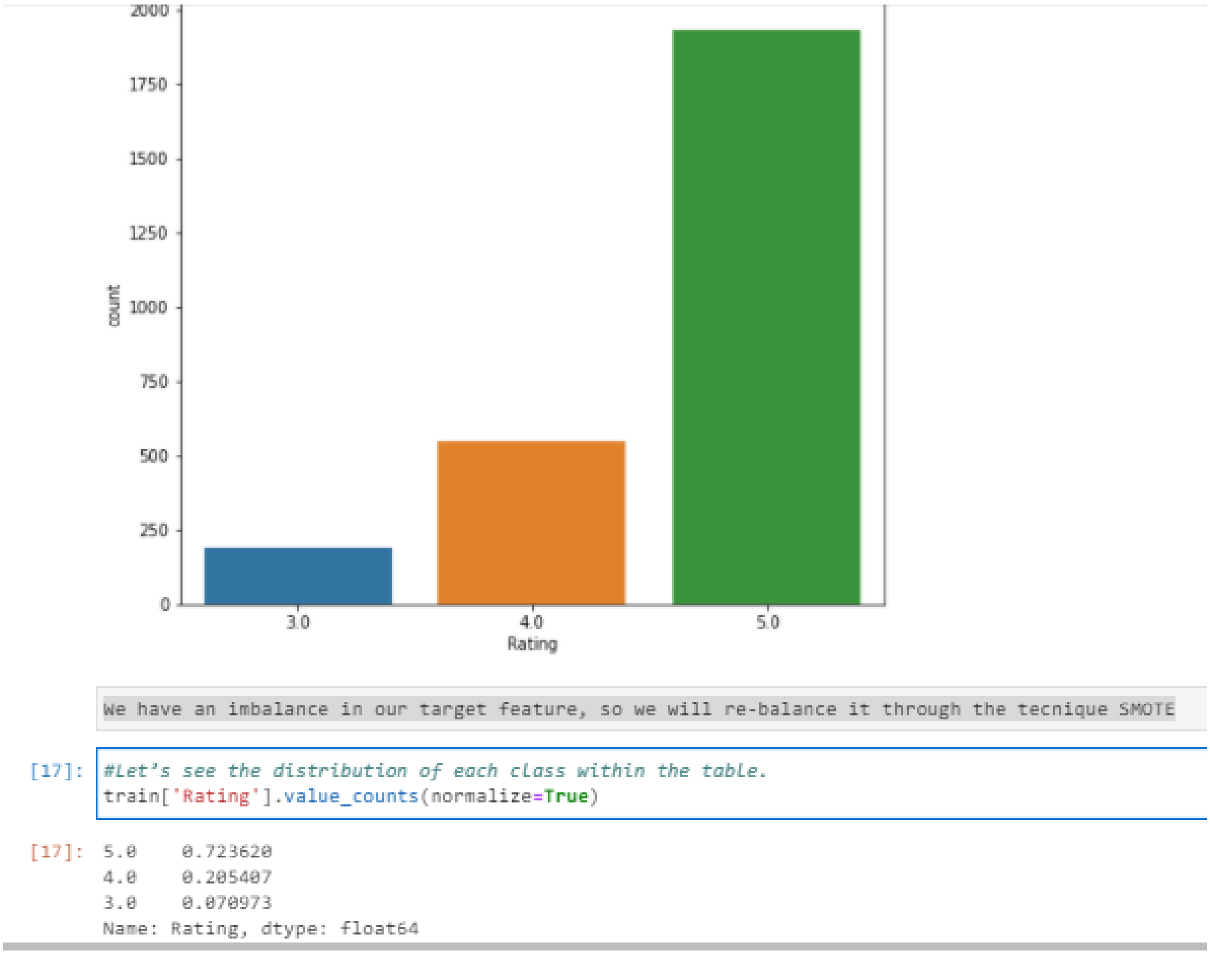
```
[10]: #we should see the similar if we plot through matplotlib:
lens = train.Full_Review.str.len()
sns.distplot(lens)
plt.title("Distribution for Lengths of Comments")
plt.show()
```

Above is a plot showing the comment length frequency. As noticed, most of the reviews are short with only a few reviews longer than 1000 words and/or 1750.



```
[13]: # checking null values through heatmap
      print(sns.heatmap(train.isnull()))

AxesSubplot(0.125,0.125;0.62x0.755)
```

#Blanks are the missing values. All 3 columns have missing data as commented previously.

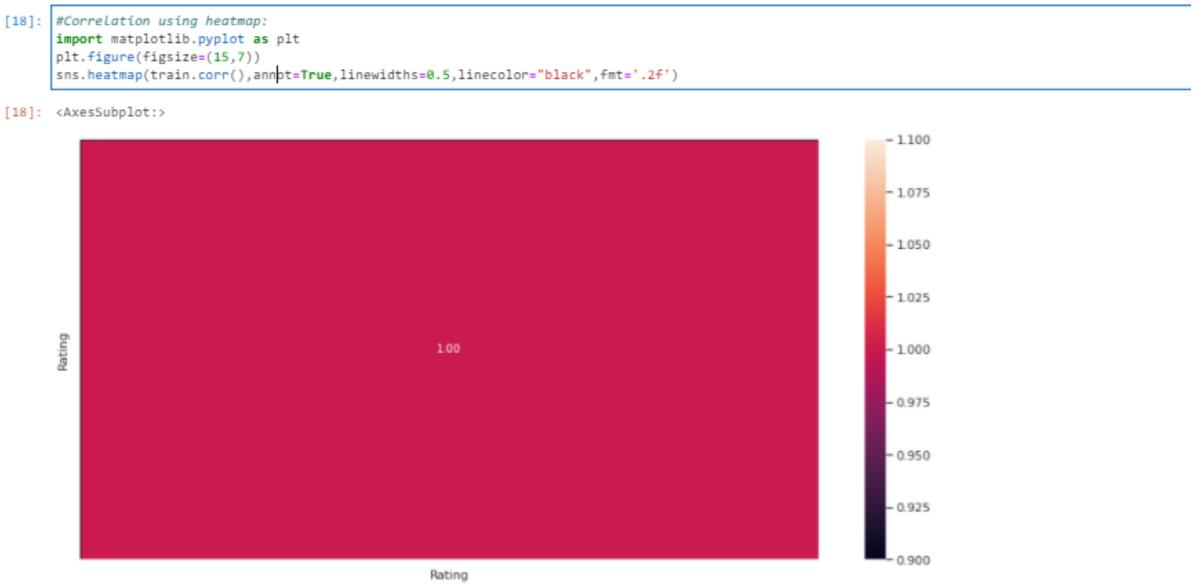


```
We have an imbalance in our target feature, so we will re-balance it through the tecnique SMOTE
```

```
[17]: #Let's see the distribution of each class within the table.
      train['Rating'].value_counts(normalize=True)

[17]: 5.0    0.723620
      4.0    0.205407
      3.0    0.070973
      Name: Rating, dtype: float64
```

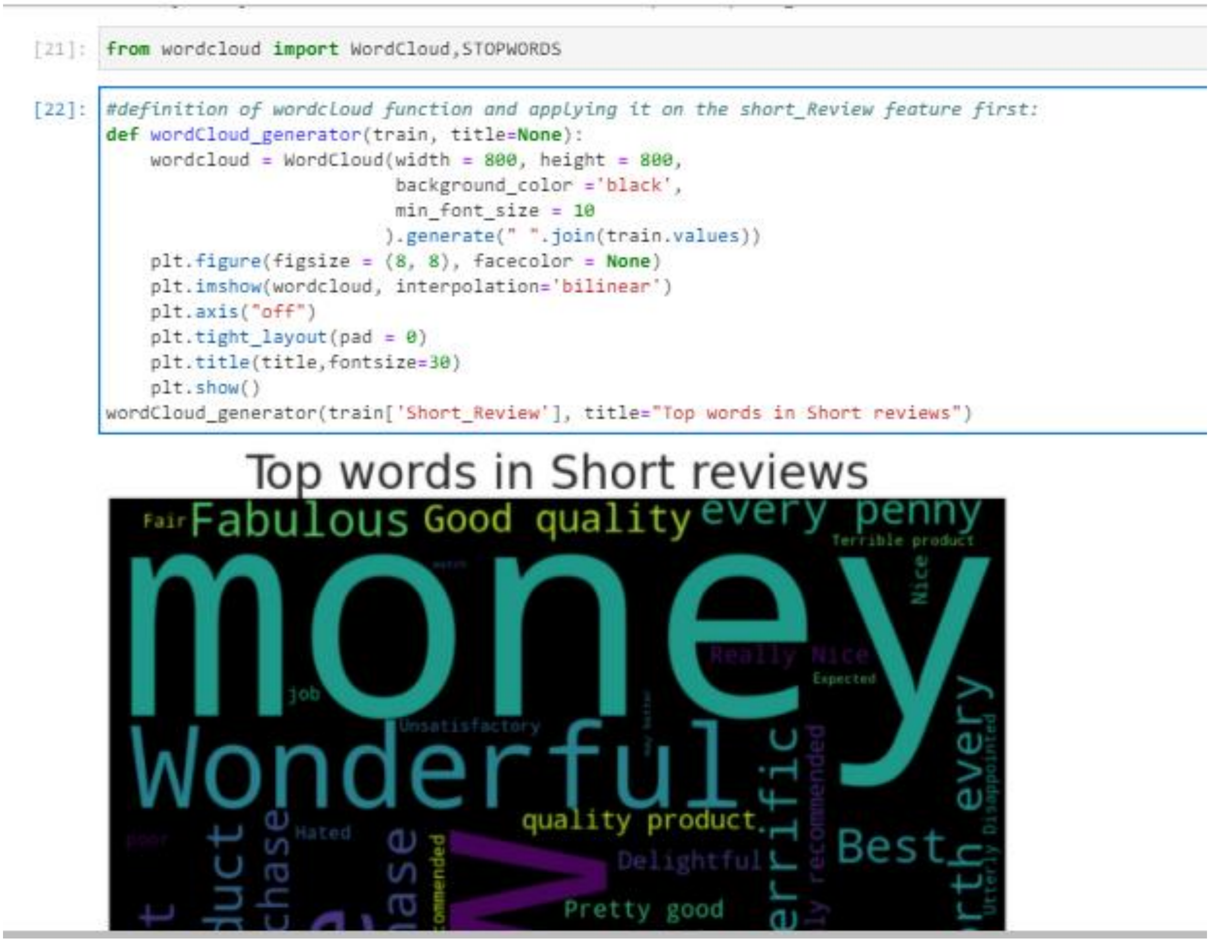
We have an imbalance in our target feature, so we will re-balance it through the tecnique SMOTE.

As we just mentioned, our target feature "Rating" is an imbalanced feature which has mostly rating number 5 and which represents 72% of the ratings. As we said, we will use SMOTE techique to adjust and remove this imbalance issue.



```
[18]: #Correlation using heatmap:
      import matplotlib.pyplot as plt
      plt.figure(figsize=(15,7))
      sns.heatmap(train.corr(),annot=True,linewidths=0.5,linecolor="black",fmt='.2f')

[18]: <AxesSubplot:>
```

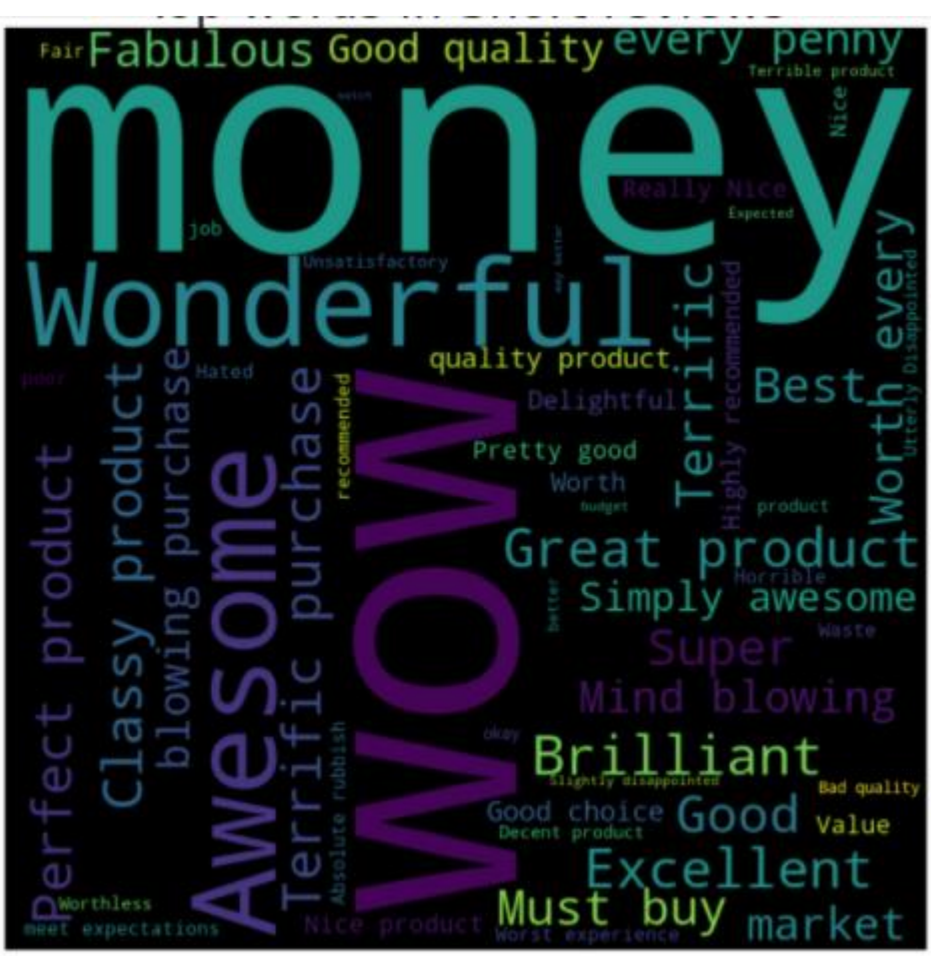
No correlation is expected between our features as we only have one numerical feature. No other numerical feature to correlate with.

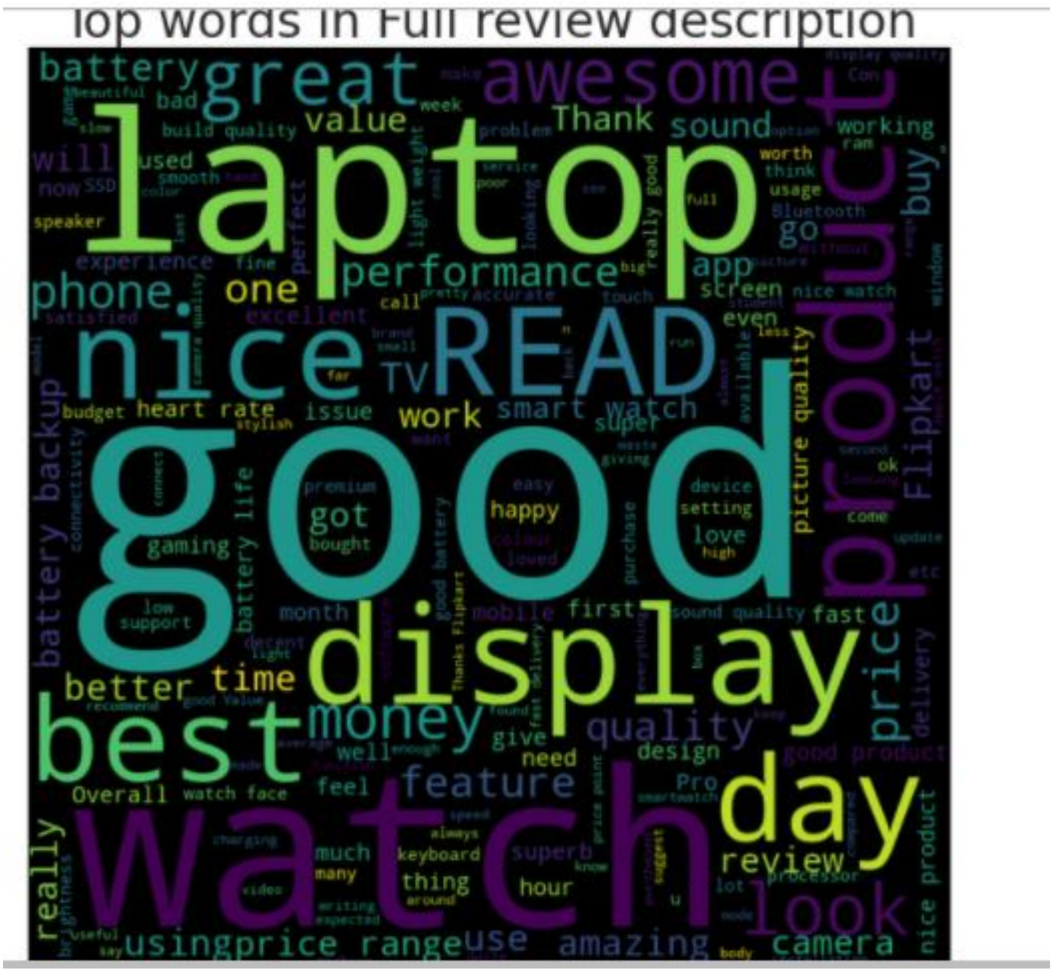


```
[21]: from wordcloud import WordCloud,STOPWORDS
```

```
[22]: #definition of wordcloud function and applying it on the short_Review feature first:
      def wordCloud_generator(train, title=None):
          wordcloud = WordCloud(width = 800, height = 800,
                                background_color ='black',
                                min_font_size = 10
                               ).generate(" ".join(train.values))
          plt.figure(figsize = (8, 8), facecolor = None)
          plt.imshow(wordcloud, interpolation='bilinear')
          plt.axis("off")
          plt.tight_layout(pad = 0)
          plt.title(title,fontsize=30)
          plt.show()
      wordCloud_generator(train['Short_Review'], title="Top words in Short reviews")
```

Top words in Short reviews


Fair
Fabulous
Good quality
every penny
Terrible product
money
Nice
Really Nice
Expected
job
Unsatisfactory
Wonderful
Terrific
Highly recommended
Worth every
Utterly Disappointed
quality product
Best
poor
Hated
Delightful
Pretty good
Worth
budget
product
Perfect product
Classy product
blowing purchase
Awesome
Terrific purchase
recommended
wow
Great product
Horrible
better
Simply awesome
Waste
Super
Mind blowing
okay
Brilliant
Slightly disappointed
Bad quality
Good choice
Good
Value
Decent product
Excellent
Absolute rubbish
Worthless
meet expectations
Nice product
Must buy
Worst experience
market



Here above you can see the words with the largest dimension has the highest influence on the label of the reviews.

- Interpretation of the Results

Give a summary of what results were interpreted from the visualizations, preprocessing and modelling.

Having target variable, we noticed we had large dataset that includes reviews in alphanumeric format which we need to process through nltk library before we add them in our machine learning training and testing.

We noticed that the training data contains 21,156 observations with 3 columns.

As we see, the Short view and Full Review are the 2 features whose data type is object. The remaining one, Rating, the target column, is the numerical one.

As seen in the non-null count, we have that all the features including the target feature have missing values as all the features have not got equal number of instances as the number of cases or entries which is equal to 21156.

From the statistic description of Rating we saw:

- saw the mean are higher than median, which means the numerical target features have a right skewed distribution.

- regarding standard distribution: Rating has low deviation. Just 0.613

- and regarding the maximum, we see 75% percentile is same as the max value, which indicates there is NO OUTLIER in our numerical feature.

And as we said, the count of values (7664) of Rating is less than the total number of rows, which means we have NULL Values in our numerical feature RATING and also in other 2 features.

As seen in the correlation matrix:

On one hand, No correlation was expected between our numerical features as we only have one numerical feature. No other numerical feature to correlate with.

On the other hand, we did expect some relation between the rating and the Short and Full Review. There must be a correlation. For that, first we need to pre-process the reviews through NLTK library and convert the reviews /strings into numerical data so that ML algorithm can be trained on it.

For Feature-engineering, we tokenized the reviews. In the tokenize function, we remove punctuations and special characters. We then lemmatize the reviews and applied TfidfVectorizer which gives us a decimal number that measures the importance of a word in any document. After pre-processing, dealing and removing unnecessary stopwords, the comment text was smaller because all stopwords were removed.

Summarizing EDA, we can say:

Basically, we tried to clean the reviews as much as possible, understand the relations and classified the reviews into ratings options we have through several classification algorithms.

Checked the mean and statistical deviation in order to get some views on the Rating distribution and also did some plotting to see visually the distribution of the Rating data.

And mostly importantly, we pre-processed and prepared the data and reviews for our machine learning through different techniques of nltk.

Approaches like cross validation and classification reports are done to analyse the results of algorithm and training performance.

We used accuracy, precision, recall, f1 score, hamming loss and log loss for validation and results evaluation and comparison.

Throughout the visualizations, we saw the following results or findings:

- at first sight we can notice we have an imbalance in our target feature, so we re-balanced it through the tecnique SMOTE

- Then, we also saw a plot showing the comment length frequency. As noticed, most of the reviews are short with only a few reviews longer than 1000 words and/or 1750.
- We also saw a plot where we could see we had the missing values in all 3 columns.
- There was another barplot where we could see an imbalance in our target feature, so we re-balanced it through the tecnique SMOTE. Our target feature "Rating" was imbalanced feature which had mostly rating number 5 and which represented 72% of the ratings. And we had to use SMOTE techique to adjust and remove this imbalance issue.
- Also, as expected, we could not see and could not compared the corelation between our features as we only had our target Rating feature as the only numerical feature. No other numerical feature to correlate with. This visualization was through seaborn’s heatmap and inserting the data train.corr() in it.
- And regarding visualizations, we also checked the most occurred words in our reviews which were Watch, good, laptop, money, wonderful, wow, awesome. These words were the words with the largest number of occurrence and has their corresponding influence on the label of the reviews.

And finally comparing the algorithms which is the one important part of the study has the following results:

The algorithms I used for the training and testing was :

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- AdaBoost Classifier
- KNeighbor Classifier

After analysing and comparing the results, we decided to go with Random Forest algorithm as it gave us the best results such as accuracy, precision, recall and therefore f1Score.

The key metrics I used to compare the results were Training and Testing accuracy, Hamming Loss and Logg Loss. I also used and compared cross validation score alongside the precision, the recall score and the f1 score.

Once checked the weights of the words that made the most occurrence through heatmap, we fitted the tf_vector on the test feature “Full_Review” and “Short_Review” and checked and predicted the ratings on test data.

We did the comparison of the results gotten through different algorithms and compared the confusion matrix and classification report.

And then finally, we saved the model as “RF.pickle” so that the technical engineer can implement wherever is needed.

CONCLUSION

- Key Findings and Conclusions of the Study

Describe the key findings, inferences, observations from the whole problem.

In this dissertation, we have studied different models to successfully predict a user’s numerical rating from its review text content. The main objective of this study was to define and select an algorithm which will give us the best results when comparing metrics such as Accuracy, Precision, Recall and F1Score alongside hamming loss and log loss.

Throughout the EDA and selection of the best algorithm we found the following main findings and conclusions that may summarize this study done on the online reviews:

- our target feature had a right skewed distribution and lower standard deviation

- no outliers were found as the maximum and 75% were the same.

- we also found our target had some imbalance data in it having

a J type distribution.

-Contrary to what one might think, implementing a text classification task in the context of imbalanced datasets was not so straightforward. Indeed, resampling techniques like SMOTE were needed in order to balance the data and alleviate biased results.

According to the evaluation metrics, the most successful classifier were RF. Moreover, using the summary of the review as well as its text content enables the classifier to better capture the discriminative power of words. Compared the algorithms we included in our study, Random Forest is the algorithm that gets the highest overall results than other algorithms.

- ## Learning Outcomes of the Study in respect of Data Science

List down your learnings obtained about the power of visualization, data cleaning and various algorithms used. You can describe which algorithm works best in which situation and what challenges you faced while working on this project and how did you overcome that.

The following steps were taken to process the data which helped me learn and understand more the process of natural language preprocessing and more:

→ Our comment_text field contains strings such as won't, didn't, etc. which contain apostrophe character('). To prevent these words from being converted to wont or didn't, the character ' represented as \' in escape sequence notation is replaced by empty character in the punctuation string.

→ Updating the list of stop words:

Stop words are those words that are frequently used in both written and verbal communication and thereby do not have either a positive or negative impact on our statement like “is, this, us, etc.”. And Single letter words if existing or created due to any preprocessing step do not convey any useful meaning and so they can be directly removed. Hence letters from b to z, will be added to the list of stop words imported directly.

→ Stemming and Lemmatizing:

The process of converting inflected/derived words to their word stem or the root form is called stemming. Many similar origin words are converted to the same word e.g. words like "stems", "stemmer", "stemming", "stemmed" as based on "stem".

On the other hand, Lemmatizing is the process of grouping together the inflected forms of a word so they can be analyzed as a single item. This is quite similar to stemming in its working but differs since it depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document.

The wordnet library in nltk will be used for this purpose. Stemmer and Lemmatizer are also imported from nltk.

→ Applying TF-IDF Vectorizer:

TF-IDF stands for Term Frequency — Inverse Document Frequency and is a statistic that aims to better define how important a word is for a document, while also taking into account the relation to other documents from the same corpus.

This is performed by looking at how many times a word appears into a document while also paying attention to how many times the same word appears in other documents in the corpus.

Stop Words: A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result.We would not want these words to take up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. You can find them in the nltk_data directory. home/pratima/nltk_data/corpora/stopwords is the directory address.

Stop words were accepted, convert to lowercase, and regular expression as its parameters. Here, we will be supplying our custom list of stop words created earlier and using lowercase option. Regular expression will have its default value.

• Limitations of this work and Scope for Future Work

Improvement → Although we have not tried several parameters in refining my model, we can do so and even find a better model which gives greater accuracy.

When finishing this project, I saw other similar projects and tried to compare the structure of study and results obtained in different projects having the same goal.

Some recommendations can be addressed in order to improve the predictive performance of our models. First of all, conducting the analysis on additional datasets would be more representative in order to validate the different results obtained in the framework of this dissertation.

Moreover, using feature selection such as selecting the set of the most popular k words within the reviews of the dataset would allow to refine the performance of the classifiers and therefore improve the accuracy metrics.

Another recommendation for further improvement is to explore the bag of opinions suggested by Qu, Ifrim and Weikum (2010). Indeed, the bag-of-words may not always be the best representation in the framework of product reviews. For instance, two reviews such as "awesome hotel in an awful town" and "awful hotel in an awesome town" are represented the same way with the bag-of-words model while they express completely opposite opinions.

In the same perspective, we can also compare the performance of unigrams, bigrams and trigrams in order to see whether it leads to improvements.

Finally, taking into account additional independent variables obtained from the text such as review length or text difficulty as well as other elements such as the helpfulness score or the identity of the reviewer could also lead to a better predictive performance.

OK.