RATINGS PREDICTION PROJECT

Submitted by:

SURAJ CHAKRABORTY

ACKNOWLEDGMENT

First and foremost, praises and thanks to the God, the Almighty, for his Shower of blessings throughout my project work to complete the project successfully.

I would like to express my deep and sincere gratitude to my SME Mr. Sajid Choudhary of Flip Robo technology, for giving me the opportunity to make this project and providing guidance on how to make the project. It was a great privilege to work under his mentorship. I would also like to thank my datatrained institute mentors for giving me all the knowledge in data science, which I am able to implement now for making the project. The projects which I had made while learning with datatrained helped me a lot in providing reference to make this project. Also the internet played a key role in helping me make the project as whenever I got stuck, I looked up on the internet for possible solutions that I could use and also to get rid of errors.

I am extremely grateful to my mother for her love, prayers and care and sacrifices for educating me and preparing me for my future.

INTRODUCTION

- ## Business Problem Framing

  Reviews and Ratings are two very important factors that give us an understanding of the product quality and decency. Product reviews are the opinions or feedbacks of customers for a particular product. Many online businesses put up a review section on their website to allow customers to rate and review the product they purchased.

  A product review helps other users get a clear idea of the product before purchasing it. They can read the reviews and get an idea on the product, and decide whether the product is worth purchasing or not. This has a great impact on the business and people tend to purchase products which have public reviews on it.

  We have a website where people write different reviews for different products and through the website people can get better understanding of the product they are looking to purchase. This will help making decision easier and people would prefer visiting the website before every purchase. But there are people who don't want to spend time in reading long reviews, for them a new feature can be added to the website i.e. rating. Ratings are the short and quick interpretation of the reviews and it generally ranges from 1 to 5. Adding the rating feature to the website in corresponding with the reviews will help the website grow more. The reviews will need to add ratings to the reviews from now. But for the old reviews we need the help of data science and machine learning to come up with a solution to add ratings to them.

- ## Conceptual Background of the Domain Problem

  We have a client who has a website where people write different reviews for technical products. Now they are adding a new feature to their website i.e. the reviewer will have to add stars (rating) as well with the review. The rating only has 5 options available 1 star,

2 stars, 3 stars, 4 stars, 5 stars. Now they want to predict ratings for the reviews which were written in the past and they don't have a rating. So, we have to build an application which can predict the rating by seeing the review.

We have to complete the project in two phases, the data collection phase and the model building phase. In the data collection phase, we have to scrape at least 20000 rows of data. In this section we need to scrape the reviews and ratings of different laptops, Phones, Headphones, smart watches, Professional Cameras, Printers, Monitors, Home theatre, Router from different ecommerce websites.

After collecting the data, you need to build a machine learning model. Before model building we need to do all data pre-processing steps involving NLP. Try different models with different hyper parameters and select the best model.

- Review of Literature

The resources provided to us by our institute datatrained, have helped me a lot in completing the project on NLP. Also the internet provides a lot of information regarding the topic. There are many articles regarding NLP in websites like towardsdatascience.com, kaggle.com, medium.com etc. Those articles gave a better understanding of the project to me and how to approach and deal with the problems.

- Motivation for the Problem Undertaken

The objective is to provide the client with the best model, which can be used to predict the ratings accurately by seeing the reviews. This will help them fulfil their target of adding the new feature to their website which eventually will help in the growth of the website.

What motivated me to take the project is the fact that the project is based on NLP, which is a newly learnt topic for me and I like to improve my knowledge on the topic, and also the fact that the

project is a full fledged project that involves both scraping the data and creating a ML model using the scraped data motivated me to take the project. Another point that I found interesting is that we had to deal with a huge amount of data in this project which definitely gave me a better understanding on dealing with large amount of data.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

  In the project we used classification models, as the target for the dataset was categorical in nature containing five unique categories. We build a model that can be used to predict the rating of a product based on a review. The project is based on NLP and in order to fit the string data into our model we had to perform vectorization for the feature and convert it to a sparse matrix. In the Machine Learning models, we had to determine the optimal values of Hyper Parameters and select the final model based on their scores.

- Data Sources and their formats

  The project consists of two phases, one for scraping the data and the other for building the model. In the first phase we had to scrape the data for different products from different websites such as amazon, flipkart, ebay, myntra etc. and hence these websites acted as the sources of data for this project. We preferred using multiple websites as the data sources as it would help our model to remove the effect of over fitting. We had to scrape the review and the rating data, and the formats of the reviews were of object datatype

that contained long written strings and the ratings were of integer datatype which contained the rating from a scale of 1 to 5.

- Data Pre-processing Done

  - First we scraped the data using a jupyter notebook and saved the dataset in an excel sheet. And then we loaded the dataset into another jupyter notebook.

  - We dropped the unwanted columns from the dataset which was the index column that had no role to play.

```
df.drop(['Unnamed: 0'], axis=1, inplace=True)
df
```

Screenshot of the code

  - Checked for any missing values in the dataset and we found some missing data. We then dropped the missing values.

```
: df.dropna(axis=0, inplace=True)
  df
```

Screenshot of the code

  - We then checked the data types of the feature and the target. And also checked the percentage ratio of all the unique target data.

```
# checking datatypes

df.dtypes
```

```
Review    object
Rating     int64
dtype: object
```

We have the rating as integer and review as object datatype.

```
print('5 star ratio= ', len(df[df['Rating']==5])/len(df['Rating'])*100,'%')
print('4 star ratio= ', len(df[df['Rating']==4])/len(df['Rating'])*100,'%')
print('3 star ratio= ', len(df[df['Rating']==3])/len(df['Rating'])*100,'%')
print('2 star ratio= ', len(df[df['Rating']==2])/len(df['Rating'])*100,'%')
print('1 star ratio= ', len(df[df['Rating']==1])/len(df['Rating'])*100,'%')
```

Screenshot of the code

- Then we created a new column that contained the length of the feature data.

```
# new column for review length

df['Length']= df.Review.str.len()
df
```

Screenshot of one of the code

- We started data cleaning by first converting all the text in the feature into lower case.

```
# converting all the reviews to lower case.

df['Review']=df['Review'].str.lower()
df
```

Screenshot of the code

Then we cleaned all the junk in the data by replacing all the punctuations, white spaces, web addresses etc by a space or a single text that defines the element such as an email id.

```
# Cleaning the data for unnecessary elements using regular expression.


# replacing email addresses with 'emailaddress'
df['Review']= df['Review'].str.replace(r'[a-zA-Z0-9.-]+@[a-zA-Z0-9-]+\.[\w]+','emailaddress')

# replacing urls with 'webaddress'
df['Review']= df['Review'].str.replace(r'(https?://)?(www\.)?(\w+)\.(\w)+','webaddress')

# replacing money symbol with string
df['Review']= df['Review'].str.replace(r'\₹|\$','currency')

# replacing number with string
df['Review']= df['Review'].str.replace(r'\d+(\.\d+)?','number')
```

```
# replacing punctuations with space
df['Review']= df['Review'].str.replace(r'[^\w\d\s]',' ')

# replacing whitespace between terms with single space
df['Review']= df['Review'].str.replace(r'\s+',' ')

# removing leading and trailing whitespace
df['Review']= df['Review'].str.replace(r'^\s{2,}|\s{2,}$','')
```

Screenshot of the code

After this we removed the stopwords from the data and before removing them we made some alterations to the English stopwords.

```
#removing stopwords

from nltk.corpus import stopwords

exclude_words = set(("didn't","doesn't","don't", "isn't", 'not'))
stop_words= set(stopwords.words('english')+['phone','laptop','monitor','camera','printer','headphone','router','product','number'
new_stop_words = stop_words - exclude_words

df['Review']= df['Review'].apply(lambda x: ' '.join(j for j in x.split() if j not in new_stop_words))
```

Screenshot of the code

- Then again we created a new column containing the new length of the cleaned data and then compared the new length to the old length.

```
# new column for clean length

df['Cleaned_length']= df.Review.str.len()
df
```

	Review	Rating	Length	Cleaned_length
0	everything works fast speed due inumber number...	5	300.0	241
1	good morning nice used day strong muscular wor...	5	332.0	194
2	flipkart took days deliver used hours first im...	5	724.0	467
3	got numberk particular price amazing went onn ...	5	836.0	561
4	best price range bought summer sale rsnumber t...	5	543.0	310
...	...	...	...	...
34379	good good bandwidth	5	28.0	19
34380	advanced version modem many additional features	5	58.0	47
34381	horribly good	5	19.0	13
34382	super	5	10.0	5
34383	good easy installed	5	35.0	19

34121 rows × 4 columns

Here we have created a new column that contains the length of the review after cleaning the data.

```
print('original length', df.Length.sum())
print('cleaned length', df.Cleaned_length.sum())
```

Screenshot of the codes

- Checked for imbalance in the dataset and found a target value containing way more records as compared to the other target values. Hence we down-sampled that target value data to that of another target value.

```
# downsampling the 5 star rating data.

x= df.drop('Rating',axis=1)
y= df.Rating

x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.25)
```

```
train= pd.concat([x_train,y_train], axis=1)
test= pd.concat([x_test,y_test], axis=1)

five= train[train.Rating==5]
one= train[train.Rating==1]
```

```
from sklearn.utils import resample

down= resample(five, replace=False, n_samples=len(one),random_state=41)

four= train[train.Rating==4]
three= train[train.Rating==3]
two= train[train.Rating==2]

df1= pd.concat([down,one, four, three, two, test])
df1
```

Screenshot of the code

- We then performed some visualization. We checked the distribution of the lengths of the data before and after the data cleaning.

```
# Checking the distribution before cleaning

fig, axs= plt.subplots(3,2,figsize= (20,15))

sns.distplot(df1[df1['Rating']==1]['Length'], bins=20, ax=axs[0,0], label='One star')
axs[0,0].set_xlabel('1 star review length')
axs[0,0].legend()

sns.distplot(df1[df1['Rating']==2]['Length'], bins=20, ax=axs[0,1], label='Two stars')
axs[0,1].set_xlabel('2 star review length')
axs[0,1].legend()

sns.distplot(df1[df1['Rating']==3]['Length'], bins=20, ax=axs[1,0], label='Three stars')
axs[1,0].set_xlabel('3 star review length')
axs[1,0].legend()

sns.distplot(df1[df1['Rating']==4]['Length'], bins=20, ax=axs[1,1], label='Four stars')
axs[1,1].set_xlabel('4 star review length')
axs[1,1].legend()

sns.distplot(df1[df1['Rating']==5]['Length'], bins=20, ax=axs[2,0], label='Five stars')
axs[2,0].set_xlabel('5 star review length')
axs[2,0].legend()

axs[2,1].set_axis_off()

plt.show()
```

Screenshot of the code

And we also visualized the word cloud for all the unique target values in the dataset.

```
# Creating word clouds

from wordcloud import WordCloud

one= df1['Review'][df1['Rating']==1]

one_cloud= WordCloud(width=700,height=500,background_color= 'white', max_words=30).generate(' '.join(one))

plt.figure(figsize=(6,8), facecolor='b')
plt.imshow(one_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

Screenshot of the code

- Finally we converted the text data in the feature to vectors using a vectorizer.

```
# convert text into vectors using tf-idf vectorizer.

from sklearn.feature_extraction.text import TfidfVectorizer

tf_vec= TfidfVectorizer()

x= tf_vec.fit_transform(df1['Review'])
y= df1['Rating']
```

Screenshot of the code

- Data Inputs- Logic- Output Relationships

  The data that we collected contained a text data column and the target column. The feature column contained string data and the target contained categorical integer data and there were certain strings in the text data that related to a particular target value in the dataset. Taking this as the logic behind, we created our models.

- Hardware and Software Requirements and Tools Used

  Hardware required: Minimum Intel i3 processor with 8 GB of RAM.

  Operating system: Windows 7, 8, or 10 or LINUX or MacOS

  Tools used:

  - Jupyter notebook: For the coding and creating the project.
  - Microsoft edge: For hosting.
  - Chrome driver: For scrapping data from websites.
  - Microsoft word: For making the project report.
  - Microsoft PowerPoint: For making the PPT.

  Libraries & Packages used:

  - Numpy: Used to perform log transformation on the dataset for treating the skewness. Also used to find the exponent of the final predicted result to inverse the logarithm and round up the values.

- Pandas: Used to load the dataset into the jupyter notebook and controlling the viewing pattern of the data.
- Selenium: Used for controlling web browser through programs and performing browser automation.
    - Webdriver: Used for initializing the web driver to a variable.
    - Exceptions: Used to deal with any unwanted exception during scraping.
- Time: Used to halt the scrapping process for a certain time to let a page load successfully.
- nltk: used for natural language processing to make machine understand human language and process it.
- Wordcloud: Used to create a word cloud highlighting the most commonly occurring words.
- Matplotlib: Uses to visualize and plot the distribution of data in columns as well as the relation between different columns.
- Seaborn: Used to visualize the distribution and correlation between data using different types of figures and graphs.
- Sklearn: Used to perform different operations using different modules in it.
    - Accuracy_score: To find the accuracy of a model performance.
    - Confusion_matrix: Used for displaying the confusion matrix formed by the model.
    - Classification_report: Used to find the recall, precision and f1 score of the models.
    - Train_test_split: To split the data in train and test data.
    - Resample: Used to perform up-sampling or down-sampling.
    - TfidfVectorizer: Used to convert text into vectors.
    - Using sklearn we also imported different models for the training and the testing of the data. E.g. LogisticRegression, MultinomialNB etc.

- Cross_val_score: Used to check the cross validation scores for different models.
- RandomizedSearchCV: Used to perform hyper-parameter tuning to find the best parameters for the models.

- Joblib: Used to save the model that we created in .pkl format.

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

  The approach that we followed is:

  - Loaded the dataset into the jupyter notebook.
  - Performed EDA to get better insights into the dataset.
  - Checked for missing values in the dataset and treated them.
  - Created a new column containing the length of the feature data.
  - Converted all the text in the feature to lower case.
  - Performed data cleaning by removing all the unwanted texts, symbols, stop-words etc.
  - Created another new column containing the new length of the cleaned data.
  - Checked for imbalance in the dataset and performed down-sampling when found some.
  - Performed some visualization by comparing the distribution of the length of data before and after data cleaning.
  - Visualized the word cloud for every unique target values.

- Converted the feature text data into vectors using a vectorizer.
- Found the best random state and preformed train test split using it.
- Trained different models that can be used for a multi class classification.
- Checked CV score for all the models for any overfitting or underfitting.
- Hyper-parameter tuned the better performing models to find the best parameters for those models.
- Trained the models using those parameters and saved the highest scoring model as the final model.

- Testing of Identified Approaches (Algorithms)

  Since the project belonged to multi-class classification, the algorithms that we used are:

  - Multinomial Naive bayes
  - Decision tree classifier
  - Random Forest Classifier
  - K- Nearest Neighbour classifier
  - Gradient Boosting Classifier

- Run and Evaluate selected models

  - Multinomial Naive bayes: Multinomial Naive Bayes classifier is a specific instance of a Naive Bayes classifier which uses a multinomial distribution for each of the features. The multinomial Naive Bayes classifier is suitable for classification with discrete features. The multinomial distribution normally requires integer feature counts. However, fractional counts such as tf-idf also work.

```
# Model creation

from sklearn.naive_bayes import MultinomialNB

mnb= MultinomialNB()
mnb.fit(x_train,y_train)
pred_mnb= mnb.predict(x_test)

print(accuracy_score(y_test,pred_mnb),'\n')
print(confusion_matrix(y_test,pred_mnb),'\n')
print(classification_report(y_test,pred_mnb),'\n')
```

```
0.49848584595128376

[[1170    9   58  251   56]
 [ 489   49  157  374   59]
 [ 257   22  242  635  134]
 [ 134   13   58 1126  479]
 [  38    2   13  571 1199]]

              precision    recall  f1-score   support

           1       0.56      0.76      0.64      1544
           2       0.52      0.04      0.08      1128
           3       0.46      0.19      0.27      1290
           4       0.38      0.62      0.47      1810
           5       0.62      0.66      0.64      1823

    accuracy                           0.50      7595
   macro avg       0.51      0.45      0.42      7595
weighted avg       0.51      0.50      0.45      7595
```

Screenshot of the code

Using Multinomial naive bayes, we found the accuracy score to be 50%, and the F1 score of be 64%, 8%, 27%, 47% and 64% for all the unique target values starting from 1 to 5.

- Decision tree classifier: Decision tree classifier builds classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed.

```
from sklearn.tree import DecisionTreeClassifier

dtc= DecisionTreeClassifier()
dtc.fit(x_train,y_train)
pred_dtc= dtc.predict(x_test)

print(accuracy_score(y_test,pred_dtc),'\n')
print(confusion_matrix(y_test,pred_dtc),'\n')
print(classification_report(y_test,pred_dtc),'\n')
```

```
0.5285055957867018

[[ 937  218  202   97   90]
 [ 230  499  186  124   89]
 [ 188  158  591  177  176]
 [ 137  116  221  822  514]
 [  82   77  149  350 1165]]

              precision    recall  f1-score   support

           1       0.60      0.61      0.60      1544
           2       0.47      0.44      0.45      1128
           3       0.44      0.46      0.45      1290
           4       0.52      0.45      0.49      1810
           5       0.57      0.64      0.60      1823

    accuracy                           0.53      7595
   macro avg       0.52      0.52      0.52      7595
weighted avg       0.53      0.53      0.53      7595
```

Screenshot of the code

Using Decision tree classifier, we found the accuracy score to be 53%, and the F1 score of be 60%, 45%, 45%, 49% and 60% for all the unique target values starting from 1 to 5.

- Random forest classifier: Random forest (RF) is an ensemble classifier that uses multiple models of several DTs to obtain a better prediction performance. It creates many classification trees and a bootstrap sample technique is used to train each tree from the set of training data.

```
from sklearn.ensemble import RandomForestClassifier

fr=RandomForestClassifier()
fr.fit(x_train,y_train)
predfr= fr.predict(x_test)

print(accuracy_score(y_test,predfr),'\n')
print(confusion_matrix(y_test,predfr),'\n')
print(classification_report(y_test,predfr),'\n')
```

```
0.5953917050691244

[[1233   99   88   53   71]
 [ 350  453  166   85   74]
 [ 206  132  575  224  153]
 [ 130   56  140  899  585]
 [  60   20   53  328 1362]]

              precision    recall  f1-score   support

           1       0.62      0.80      0.70      1544
           2       0.60      0.40      0.48      1128
           3       0.56      0.45      0.50      1290
           4       0.57      0.50      0.53      1810
           5       0.61      0.75      0.67      1823

    accuracy                           0.60      7595
   macro avg       0.59      0.58      0.58      7595
weighted avg       0.59      0.60      0.58      7595
```

Screenshot of the code

Using Random forest classifier, we found the accuracy score to be 59%, and the F1 score of be 70%, 48%, 50%, 53% and 67% for all the unique target values starting from 1 to 5.

- K- Nearest Neighbour classifier: K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new data-points which further means that the new data point will be assigned a value based on how closely it matches the points in the training set.

```
from sklearn.neighbors import KNeighborsClassifier

knn= KNeighborsClassifier()
knn.fit(x_train,y_train)
predknn= knn.predict(x_test)

print(accuracy_score(y_test,predknn),'\n')
print(confusion_matrix(y_test,predknn),'\n')
print(classification_report(y_test,predknn),'\n')
```

```
0.3377221856484529

[[233  73 237 476 525]
 [ 67 181 191 342 347]
 [ 44  53 452 366 375]
 [ 25  38 354 773 620]
 [ 16  34 274 573 926]]

              precision    recall  f1-score   support

           1       0.61      0.15      0.24      1544
           2       0.48      0.16      0.24      1128
           3       0.30      0.35      0.32      1290
           4       0.31      0.43      0.36      1810
           5       0.33      0.51      0.40      1823

    accuracy                           0.34      7595
   macro avg       0.40      0.32      0.31      7595
weighted avg       0.40      0.34      0.32      7595
```

Screenshot of the code

Using K Neighbour classifier, we found the accuracy score to be 34%, and the F1 score of be 24%, 24%, 32%, 36% and 40% for all the unique target values starting from 1 to 5.

- Gradient Boosting classifier: Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model in order to minimize the error.

```
from sklearn.ensemble import GradientBoostingClassifier

gdb= GradientBoostingClassifier()
gdb.fit(x_train,y_train)
predgdb= gdb.predict(x_test)

print(accuracy_score(y_test,predgdb),'\n')
print(confusion_matrix(y_test,predgdb),'\n')
print(classification_report(y_test,predgdb))
```

```
0.48321263989466756

[[ 999  135  128   98  184]
 [ 340  234  205  155  194]
 [ 193  143  374  303  277]
 [  96   61  177  775  701]
 [  42   29   59  405 1288]]

              precision    recall  f1-score   support

           1       0.60      0.65      0.62      1544
           2       0.39      0.21      0.27      1128
           3       0.40      0.29      0.33      1290
           4       0.45      0.43      0.44      1810
           5       0.49      0.71      0.58      1823

    accuracy                           0.48      7595
   macro avg       0.46      0.46      0.45      7595
weighted avg       0.47      0.48      0.47      7595
```

Screenshot of the code

Using Gradient boosting classifier, we found the accuracy score to be 48%, and the F1 score of be 62%, 27%, 33%, 44% and 58% for all the unique target values starting from 1 to 5.

- Key Metrics for success in solving problem under consideration

Key metrics that we used are:

- Accuracy Score: Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. In multi-label classification, this function computes subset accuracy, the set of labels

predicted for a sample must exactly match the corresponding set of labels in the target sample. The more the match is, the higher is the accuracy and vice versa.

- Confusion Matrix: A confusion matrix is a tabular summary of the number of correct and incorrect predictions made by a classifier. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. It can be used to evaluate the performance of a classification model. Also it gives us insight not only into the errors being made by the classifier but more importantly the types of errors that are being made.

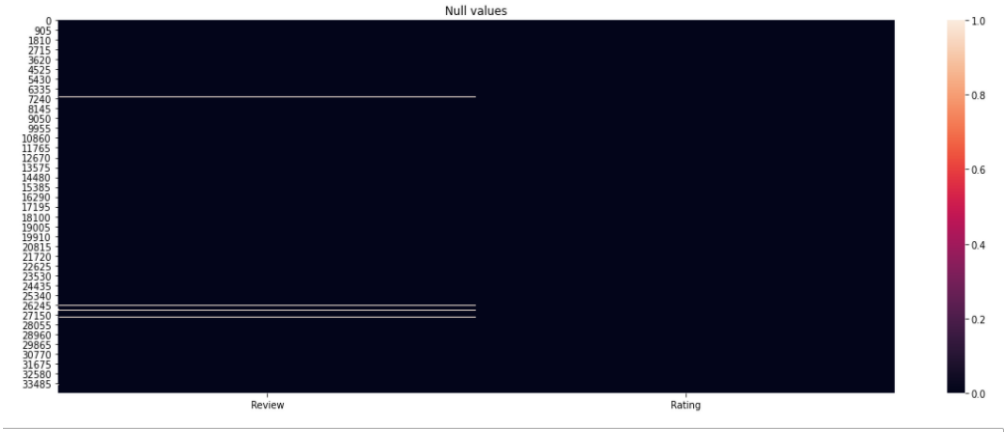
- Classification Report: A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report. The main classification metrics are precision, recall and f1-score.
  Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true and false positives.
  Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.
  The f1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, f1 scores are lower than accuracy measures as they embed precision and recall into their computation.

- Visualizations

  The plots that are used are:

  - Heatmap:

```
plt.figure(figsize=[20,7])
sns.heatmap(df.isnull())
plt.title('Null values')
plt.show()
```



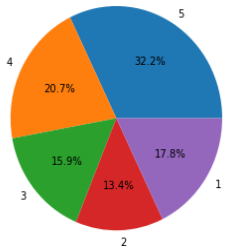
Screenshot of the plot

Heat map was used for checking any missing values in the dataset. We found that some missing values were present in it and then treated those missing values.

  - Pie Chart:

```
data = [len(df[df['Rating']==5])/len(df['Rating'])*100, len(df[df['Rating']==4])/len(df['Rating'])*100,
       len(df[df['Rating']==3])/len(df['Rating'])*100, len(df[df['Rating']==2])/len(df['Rating'])*100,
       len(df[df['Rating']==1])/len(df['Rating'])*100]
label= [5,4,3,2,1]

plt.pie(data, labels= label, autopct='%1.1f%%')
plt.tight_layout()
plt.show()
```



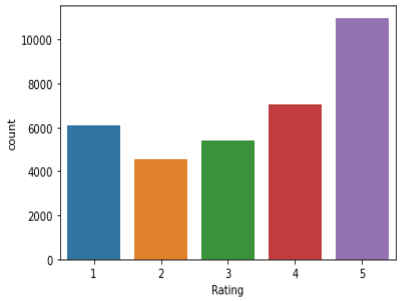
Screenshot of the plot

We used the Pie chart to visualize the percentage ratio of all the unique target values in the dataset. We found the target 5 star rating having the highest percent of data, followed by 4

star rating and the 2 star rating gave us the least percentage of data.

- Count Plot:

```
sns.countplot(df['Rating']);
```

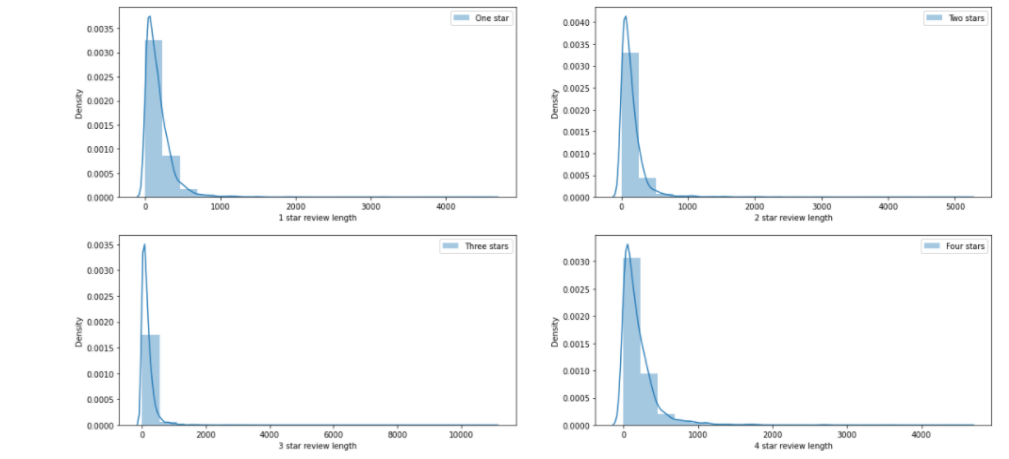


Screenshot of the plot

We used the count plot to check of imbalance in the dataset. And we found that there was some imbalance present in the dataset, the 5 star rating contained much more records as compared to other ratings in the dataset.

- Distribution plot:

```
plt.show()
```



Screenshot of the plot

The distribution plot was used two times in the project. It was used to see the distribution of the length of data for all unique target values before and after data cleansing using subplots. By comparing the two plots, we found that after

performing data cleaning, the skewness in the length was reduced.

- Word cloud:

```
four= df1['Review'][df1['Rating']==4]

four_cloud= WordCloud(width=700,height=500, max_words=30).generate(' '.join(four))

plt.figure(figsize=(6,8), facecolor='b')
plt.imshow(four_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

Screenshot of the plot

We used the word cloud to visualize the mostly occurring words for all the unique target values in the dataset. We displayed the word cloud for all the ratings and the words that occurred the most were highlighted in the display.

- Interpretation of the Results

From the pre-processing we interpreted:

- The dataset was fairly big with respect to the number of records in it.
- An unwanted column was present in the dataset, which had no role to play in building the model.
- The dataset contained multiple target values, which meant that this was a multi-class classification problem.
- There were some missing values in the dataset.
- The dataset was imbalanced, with 5 star ratings data being much higher in number as compared to others.
- The data cleaning is a very important part in a NLP project.

- After the data cleansing, few null values we found in the data, which meant those data contained only junk values.

From the visualization we interpreted:

- The 5 stars target value had the highest percentage of data followed by 4 stars, while 2 stars had the least percentage of data.
- After cleansing the data, the skewness in the distribution of the length of data in all the unique target values reduces drastically.
- The word ‘good’ was in majority in all the unique target data in the dataset.

From the modelling we interpreted:

- Finding the random state that performs the best can be very useful in getting a good score.
- Since this was a multi-class classification problem, we had to use specific models for it.
- Building multiple models helps us to find and choose the best fit model for our dataset.
- Performing cross validation test is very important to detect if there is any overfitting or underfitting in the training.
- The model that has the least difference between the accuracy score and the CV score can be considered the best performing model.
- Hyper parameter tuning plays a crucial role in finding the best parameters for the model and selecting a final model.
- In this project the Random forest classifier performed the best among all the models.

CONCLUSION

- ## Key Findings and Conclusions of the Study

  We found that while scraping data, it is important to try fetching the data for all particular target values. This will help us during our model building phase. We fetched nearly 35000 records, which formed a large dataset. We found that the project was a multi class classification project. We had an index column in the dataset, which did not have any role to play in the modelling, also some missing values were found in the review data of the dataset. We also understood that in a NLP project, the cleansing of the data play a very crucial role in the model performance. After cleaning the data, we found few missing values, which meant that those records contained only junk data. We found the word 'good' being highlighted in each of the word-clouds for every particular rating, that might affect the model, and hence we had to remove the word from the dataset. We found that we could use only certain specific models to train the data as it was a multi-class classification problem. The checking of cross validation score and performing the hyper parameter tuning helped us in the selection of the final model for this project.

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

  While scraping the data, we used many websites, but there were more websites which we could not use, as no reviews were present there. Also we tried but could not use the website 'snapdeal' as it did not contain the rating data in text.

  From this project we learnt that EDA and Data cleaning play a very crucial role in the outcome of a NLP project. The EDA when done thoroughly gives us many insights into the data. We also learnt that having a balanced dataset for training a classification model is very important in order to get a good performance. From the data

cleaning, we got an idea on what type of data we need to avoid while working on NLP and how to do alternations with stopwords, we also learnt that the word cloud is a very important feature, this helps us identify the mostly occurring words in the data and lets us check if there is any common occurring word for multiple targets, which can affect the training. We got the understanding on which models we can use while working on a multi class classification problem and how cross validation and hyper parameter tuning can help us find the best model for out project.

- Limitations of this work and Scope for Future Work

  The model we build was built by taking the reviews of certain products from several websites, but since the type of reviews may vary for different products, the model may perform slightly weak in case of reviews of products other than those we used for training.

  The model that we built can be used by the client to predict ratings for the reviews which were written in the past and for which they don't have a rating. And they can now add the new rating feature to their website, which potentially will attract more viewers to the website.

  To further improve the results, more and more data of various products can be fetched from different websites to train the model. Also more number of models can be used to check which model performs best. And for tuning the models we can use more number of parameters for the score to increase.