RATINGS PREDICTION PROJECT

SUBMITTED BY:

SHUBHAM SHUKLA

ACKNOWLEDGMENT

I would like to thank Flip Robo Technologies for providing me with the opportunity to work on this project from which I have learned a lot. Some of the reference sources are as follows:

1. www.google.com
2. www.analyticsvidhya.com
3. www.stackoverflow.com

INTRODUCTION

BUSINESS PROBLEM FRAMING

- This is a Machine Learning Project performed on customer reviews. Reviews are processed using common NLP techniques.

- Millions of people use **Amazon and Flipkart** to buy products. For every product, people can rate and write a review. If a product is good, it gets a positive review and gets a higher star rating, similarly, if a product is bad, it gets a negative review and lower star rating. My aim in this project is to predict star rating automatically based on the product review.

- The range of star rating is 1 to 5. That means if the product review is negative, then it will get low star rating (possibly 1 or 2), if the product is average then it will get medium star rating (possibly 3), and if the product is good, then it will get higher star rating (possibly 4 or 5).

- This task is similar to Sentiment Analysis, but instead of predicting the positive and negative sentiment (sometimes neutral also), here I need to predict the star rating.

AIM OF THIS PROJECT

Our goal is to make a system that automatically detects the starrating based on the review.

CONCEPTUAL BACKGROUND OF THEDOMAIN PROBLEM

- The advent of electronic commerce with growth in internet and network technologies has led customers to move to online retail platforms such as Amazon, Walmart, Flip Kart, etc. People often rely on customer reviews of products before they buy online. These reviews are often rich in information describing the product. Customers often choose to compare between various products and brands based on whether an item has a positive or negative review. More often, these reviews act as a feedback mechanism for the seller. Through this medium, sellers strategize their future sales and product improvement.

- There is a client who has a website where people write different reviews for technical products. Now they want to add a new feature to their website i.e. The reviewer will have to add stars (rating) as well with the review. The rating is out 5 stars and it 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.

REVIEW OF LITERATURE

- This project is more about exploration, feature engineering and classification that can be done on this data. Since we scrape huge amount of data that includes five stars rating, we can do better data exploration and derive some interesting features using the available columns.

- We can categorize the ratings as:1.0,
  2.0, 3.0, 4.0 and 5.0 stars

- The goal of this project is to build an application which can predict the rating by seeing the review. In the long term, this would allow people to better explain and reviewing their purchase with each other in this increasingly digital world.

MOTIVATION OF THE PROBLEM UNDERTAKEN

- Every day we come across various products in our lives, on the digital medium we swipe across hundreds of product choices under one category. It will be tedious for the customer to make selection. Here comes 'reviews' where customers who have already got that product leave a rating after using them and brief their experience by giving reviews.

- As we know ratings can be easily sorted and judged whether a product is good or bad. But when it comes to sentence reviews, we need to read through every line to make sure the review conveys a positive or negative sense. In the era of artificial intelligence, things like that have got easy with the Natural Language Processing (NLP) technology. Therefore, it is important to minimize the number of false positives our model produces, to encourage all constructive conversation.

- Our model also provides beneficence for the platform hosts as it replaces the need to manually moderate discussions, saving time and resources. Employing a machine learning model to predict ratings promotes easier way to distinguish between products qualities, costs and many other features.

ANALYTICAL PROBLEM FRAMING

MATHEMATICAL/ANALYTICAL MODELLINGOF THE PROBLEM

- In our scrapped dataset, our target variable "**Rating** " is a **categorical** variable i.e., it can be classified as ‘1.0’, ‘2.0’,’3.0’,’4.0’,’5.0’. Therefore, we will be handling this modelling problem as classification.

- This project is done in two parts:
  - Data Collection Phase
  - Model Building Phase

Data Collection Phase:

- You have to scrape at least 20000 rows of data. You can scrape more data as well, it’s up to you. More the data better the model.

- In this section you need to scrape the reviews of different laptops, Phones, Headphones, smart watches, Professional Cameras, Printers, monitors, home theatre, and router from different e-commerce websites.

- Basically, we need these columns-
  1) Reviews of the product.
  2) Rating of the product.

- Fetch an equal number of reviews for each rating, for example if you are fetching 10000 reviews then all ratings 1,2,3,4,5 should be 2000. It will balance our data set.

- Convert all the ratings to their round number, as there are only 5 options for rating i.e., 1, 2,3,4,5. If a rating is 4.5 convert it 5.

Model Building Phase:

After collecting the data, you need to build a machine learning model. Before model building do all data pre-processing steps involving NLP. Try different models with different hyper parameters and select the best model. Follow the complete life cycle of data science. Include all the steps like-

1. Data Cleaning
2. Exploratory Data Analysis
3. Data Pre-processing
4. Model Building
5. Model Evaluation
6. Selecting the best model

DATA SOURCES AND THEIR FORMATS

- We collected the data from difference e-commerce websites like Amazon and Flipkart. The data is scrapped using Web scraping technique and the framework used is Selenium.

- We scrapped nearly 50000 of the reviews data and saved each data in a separate data frame

- In the end, we combined all the data frames into a single data frame and it looks like as follows:

```
#Combining all dataframes into a single dataframe
ratings_data=headphones.append([laptops,camera,phones,f_phones,f_poco,router],ignore_index=True)
ratings_data
```

	Product_Review	Ratings
0	It has great sound quality and bass but after ...	1.0
1	[BIG UPDATE]IT BROKE ALONG THE RIGHT HINGE, I ...	2.0
2	This is a premium quality product from boAt-Li...	5.0
3	The boAt Rockerz line is boAt's super cheap li...	1.0
4	These are super comfortable and premium lookin...	5.0
...	...	...
49995	Honest rating for this product is 5/5 . Just f...	4
49996	I have bought 3 of these little guys, for my s...	5
49997	Its a great product. Works really fine. It wil...	4
49998	Service is very good , I would say to keep up ...	5
49999	Just buy it if u need bigger coverage buy a re...	5

50000 rows × 2 columns

- Then, we will save this data in a csv file, so that we can dothe pre-processing and model building.

DATA PRE-PROCESSING

- Handling missing data using fillna and checking the data types.

```
#Checking for null values
df.isnull().sum()
```

```
Product_Review    80
Ratings            0
dtype: int64
```

```
#We can handle missing data by filling them with 'No Review' using fillna()
df['Product_Review'].fillna('No review',inplace=True)
```

```
df.isnull().sum()  #Checking after filling them
```

```
Product_Review    0
Ratings           0
dtype: int64
```

```
df.info()   #Checking the datatype of all the columns present
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Product_Review  50000 non-null  object
 1   Ratings         50000 non-null  float64
dtypes: float64(1), object(1)
memory usage: 781.4+ KB
```

- Checking average rating and value counts of each rating present

```
#Checking the average rating given by the users
avg = df['Ratings'].mean()
Avg = round(avg,1)
print("Average rating given by users is " + str(Avg))
```

```
Average rating given by users is 3.7
```

```
#Checking the value counts of the rating
df['Ratings'].value_counts()
```

```
5.0    24506
1.0    11232
4.0     8406
3.0     3680
2.0     2176
Name: Ratings, dtype: int64
```

Pre-processing using Natural Language Processing (NLP):

- We cleaned the data using regex, matching patterns in the comments and replacing them with more organized counterparts. Cleaner data leads to a more efficient model and higher accuracy. Following steps are involved:

1. Removing Punctuations and other special characters
2. Splitting the comments into individual words
3. Removing Stop Words

- There is a corpus of stop-words, that are high-frequency words such as "the", "to" and "also", and that we sometimes want to litter out of a document before further processing. Stop-words usually have little lexical content, don’t alter the general meaning of a sentence and their presence in a text fails to distinguish it from other texts. We used the one from Natural Language Toolkit a leading platform for building Python programs to work with human language.

- The code is attached below:

```
def clean_text(df, df_column_name):

    #Converting all messages to lowercase
    df[df_column_name] = df[df_column_name].str.lower()

    #Replace email addresses with 'email'
    df[df_column_name] = df[df_column_name].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$','emailaddress')

    #Replace URLs with 'webaddress'
    df[df_column_name] = df[df_column_name].str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$','webaddress')

    #Replace money symbols with 'dollars' (£ can by typed with ALT key + 156)
    df[df_column_name] = df[df_column_name].str.replace(r'£|\$', 'dollars')

    #Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes) with 'phonenumber'
    df[df_column_name] = df[df_column_name].str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$','phonenumber')

    #Replace numbers with 'numbr'
    df[df_column_name] = df[df_column_name].str.replace(r'\d+(\.\d+)?', 'numbr')

    #Remove punctuation
    df[df_column_name] = df[df_column_name].str.replace(r'[^\w\d\s]', ' ')

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

    #Remove leading and trailing whitespace
    df[df_column_name] = df[df_column_name].str.replace(r'^\s+|\s+?$', '')

    #Remove stopwords
    stop_words = set(stopwords.words('english') + ['u', 'ü', 'â', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure'])
    df[df_column_name] = df[df_column_name].apply(lambda x: ' '.join(term for term in x.split() if term not in stop_words))
```

- Tokenizing the data using RegexpTokenizer.

```
#Calling the class
clean_text(df, 'Product_Review')
df['Product_Review'].tail(3)
```

```
49997    great product works really fine usually numbr ...
49998    service good would say keep good work goin fli...
49999    buy need bigger coverage buy repeater router c...
Name: Product_Review, dtype: object
```

```
#Tokenizing the data using RegexpTokenizer
from nltk.tokenize import RegexpTokenizer
tokenizer=RegexpTokenizer(r'\w+')
df['Product_Review'] = df['Product_Review'].apply(lambda x: tokenizer.tokenize(x.lower()))
df.head()
```

	Product_Review	Ratings
0	[great, sound, quality, bass, numbr, months, u...	1.0
1	[big, update, broke, along, right, hinge, wear...	2.0
2	[premium, quality, product, boat, lifestyle, p...	5.0
3	[boat, rockerz, line, boat, super, cheap, line...	1.0
4	[super, comfortable, premium, looking, headpho...	5.0

Stemming and Lemmatizing:

• **Stemming** is the process of converting inflected/derived words to their word stem or the root form. Basically, a large number of similar origin words are converted to the same word. E.g., words like "stems", "stemmer", "stemming", "stemmed" as based on "stem". This helps in achieving the training process with a better accuracy.

• **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 **word net library in nltk** will be used for this purpose. Stemmer and Lemmatizer are also imported from nltk.

```
# Lemmatizing and then Stemming with Snowball to get root words and further reducing characters
stemmer = SnowballStemmer("english")
import gensim
def lemmatize_stemming(text):
    return stemmer.stem(WordNetLemmatizer().lemmatize(text,pos='v'))

#Tokenize and Lemmatize
def preprocess(text):
    result=[]
    for token in text:
        if len(token)>=3:
            result.append(lemmatize_stemming(token))

    return result
```

- Processing the review and assigning the updated reviewin the data frame

```
#Processing review with above Function
processed_review = []

for doc in df.Product_Review:
    processed_review.append(preprocess(doc))

print(len(processed_review))
processed_review[:3]
```

```
50000

[['great',
  'sound',
  'qualiti',
  'bass',
  'numbr',
  'month',
  'use',
  'get',
  'break',
  'without',
  'fall',
  'jerk',
  'wear',
  'normal',
  'get',
  'break',
  'one',
  'side',
```

```
df['clean_review']=processed_review  #Assigning this to the dataframe
df.head()
```

df['Product_Review'] = df['clean_review'].apply(lambda x:' '.join(y for y in x))

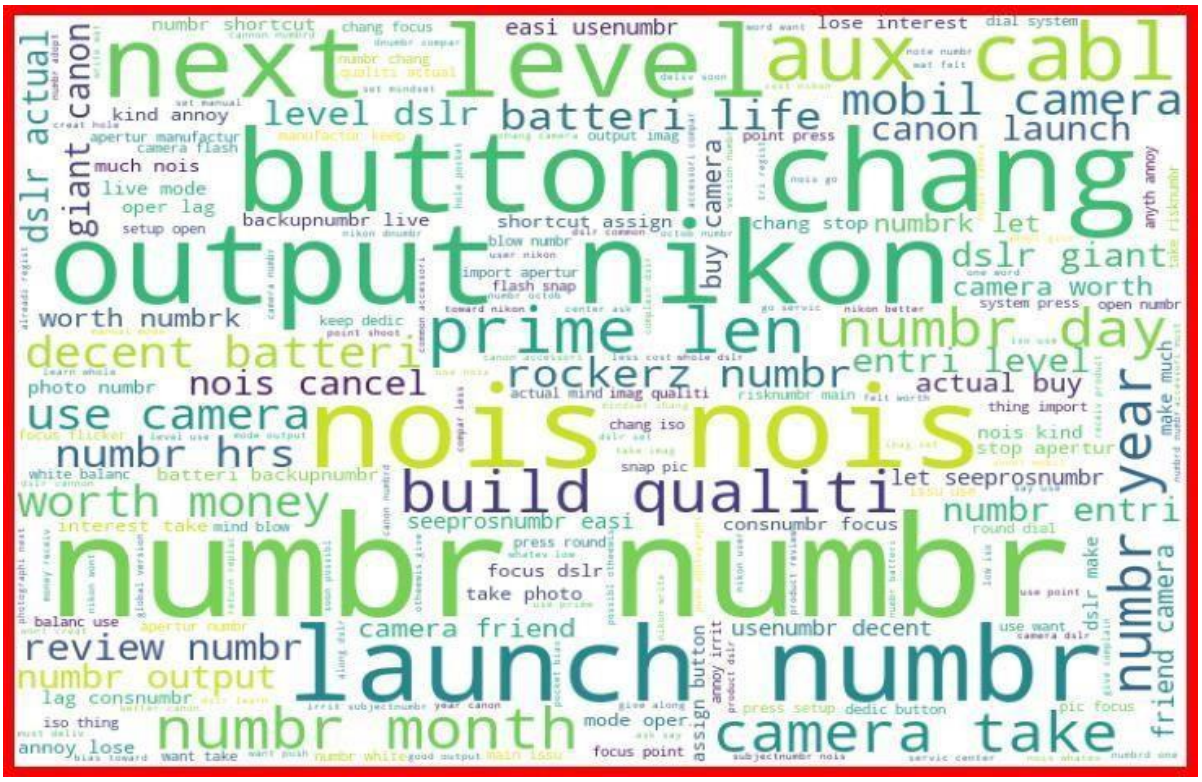
- Getting sense of words for all ratings using WordCloud

Word Cloud is a data visualization technique used for representing text data in which the size of each **word** indicates its frequency or importance.

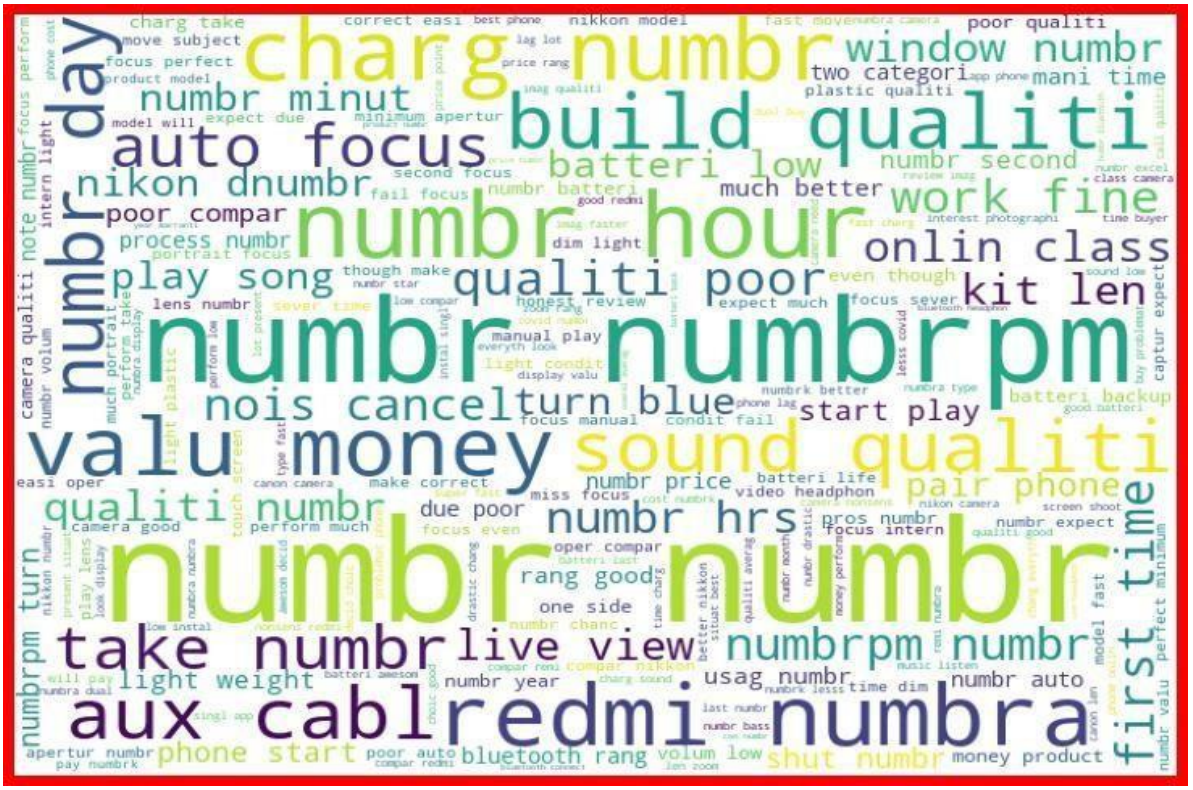
```
#Getting sense of words in Rating 1
one = df['Product_Review'][df['Ratings']==1.0]

one_cloud = WordCloud(width=700,height=500,background_color='white',max_words=200).generate(' '.join(one))

plt.figure(figsize=(10,8),facecolor='r')
plt.imshow(one_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

Similarly, we found the sense of words for ratings 2.0 – 5.0 and the output will be as follows:

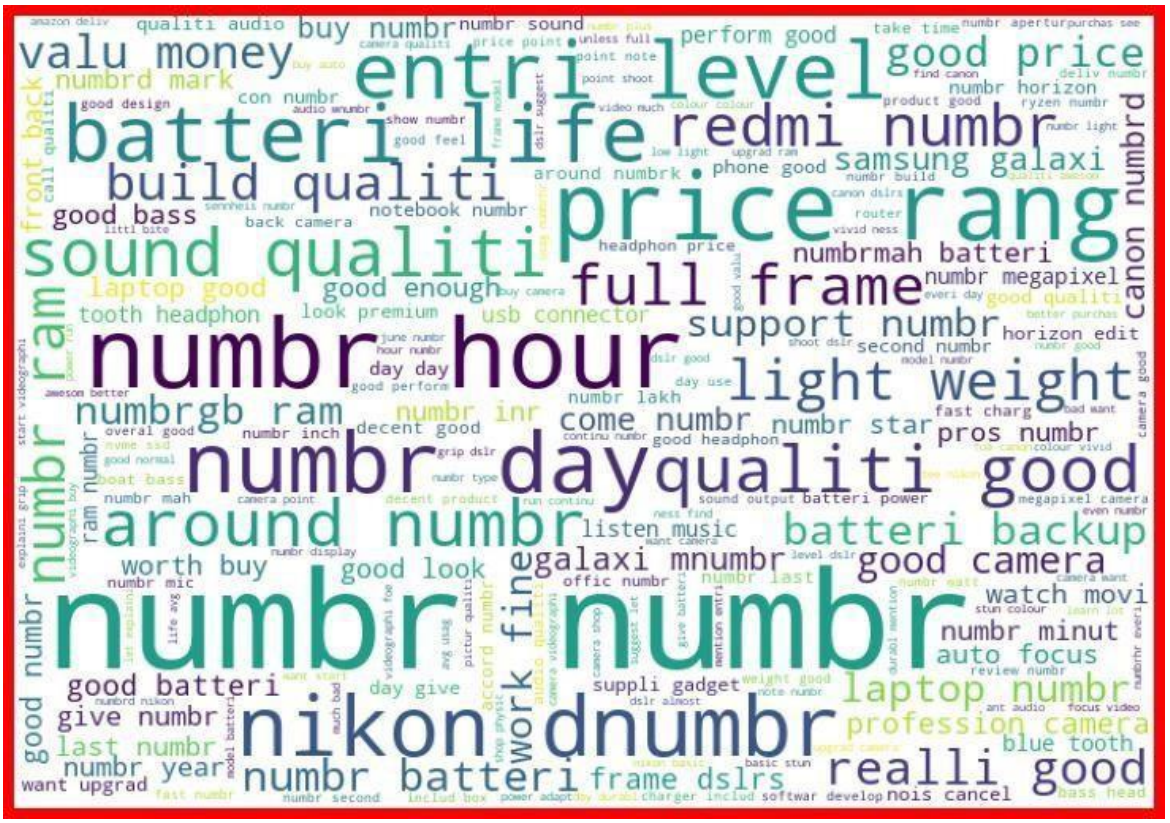
For rating 2.0:



For rating 3.0:



For rating 4.0:



For rating 5.0:

Observations:

-> The enlarged texts are the most number of words used thereand small texts are the less number of words used.

-> It varies according to the ratings.

Feature Extraction:

Here we can finally convert our text to numeric using Tf-idf Vectorizer.

Term Frequency Inverse Document Frequency (TF-IDF): This is very common algorithm to transform text into a meaningful representation of numbers which is used to fitmachine algorithm for prediction.

```
#Converting text into numeric using TfidfVectorizer
#create object
tf = TfidfVectorizer()

#fitting
features = tf.fit_transform(df['Product_Review'])
x=features
y=df[['Ratings']]
```

```
x.shape
```

```
(50000, 5828)
```

```
y.shape
```

```
(50000, 1)
```

HARDWARE AND SOFTWARE REQUIREMENTS ANDTOOLS USED

- For doing this project, the hardware used is a laptop with high end specification and a stable internet connection. While coming to software part, I had used anaconda navigator and in that I have used **Jupyter notebook** to do my python programming and analysis.

- For using a CSV file, Microsoft excel is needed. In Jupyter notebook, I had used lots of python libraries to carry out this project and I have mentioned below with proper justification:

Libraries Used:

```
#Libraries used for pre-processing
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#libraries for NLP
import re # for regex
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, RegexpTokenizer
from nltk.stem import SnowballStemmer, WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from wordcloud import WordCloud

#Libraries for model building
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

#Saving the model
import pickle

#Importing warning library to avoid any warnings
import warnings
warnings.filterwarnings('ignore')
```

MODEL/S DEVELOPMENT AND EVALUATION

- Listing down all the algorithms used for the training and testing.

```
#Initializing the instance of the model
LR=LogisticRegression()
mnb=MultinomialNB()
dtc=DecisionTreeClassifier()
knc=KNeighborsClassifier()
rfc=RandomForestClassifier()
abc=AdaBoostClassifier()
gbc=GradientBoostingClassifier()
```

```
models= []
models.append(('Logistic Regression',LR))
models.append(('MultinomialNB',mnb))
models.append(('DecisionTreeClassifier',dtc))
models.append(('KNeighborsClassifier',knc))
models.append(('RandomForestClassifier',rfc))
models.append(('AdaBoostClassifier',abc))
models.append(('GradientBoostingClassifier',gbc))
```

- Running and evaluating the models

```
Model=[]
score=[]
cvs=[]
rocscore=[]
for name,model in models:
    print('****************************',name,'*************************')
    print('\n')
    Model.append(name)
    model.fit(x_train,y_train)
    print(model)
    pre=model.predict(x_test)
    print('\n')
    AS=accuracy_score(y_test,pre)
    print('accuracy_score: ',AS)
    score.append(AS*100)
    print('\n')
    sc=cross_val_score(model,x,y,cv=5,scoring='accuracy').mean()
    print('cross_val_score: ',sc)
    cvs.append(sc*100)
    print('\n')
    print('Classification report:\n ')
    print(classification_report(y_test,pre))
    print('\n')
    print('Confusion matrix: \n')
    cm=confusion_matrix(y_test,pre)
    print(cm)
    print('\n\n\n')
```

```
#Finalizing the result
result=pd.DataFrame({'Model':Model, 'Accuracy_score': score,'Cross_val_score':cvs})
result
```

	Model	Accuracy_score	Cross_val_score
0	Logistic Regression	90.96	57.920
1	MultinomialNB	87.69	59.308
2	DecisionTreeClassifier	89.90	53.062
3	KNeighborsClassifier	89.39	48.930
4	RandomForestClassifier	91.45	56.348
5	AdaBoostClassifier	60.04	52.334
6	GradientBoostingClassifier	90.60	60.782

Key Metrics for success in solving problem under consideration

The key metrics used here were accuracy score, cross_val_score, classification report, and confusion matrix. We tried to find out the best parameters and also to increase our scores by using Hyper parameter Tuning and we will be using GridSearchCV method.

1. Cross Validation:

Cross-validation helps to find out the over fitting and under fitting of the model. In the cross validation the model is made to run on different subsets of the dataset which will get multiple measures of the model. If we take 5 folds, the data will be divided into 5 pieces where each part being 20% of full dataset. While running the Cross-validation the 1st part (20%) of the 5 parts will be kept out as a holdout set for validation and everything else is used for training data. This way we will get the first estimate of the model quality of the dataset.

In the similar way further iterations are made for the second 20% of the dataset is held as a holdout set and remaining 4 parts are used for training data during process. This way we will get the second estimate of the model quality of the dataset. These steps are repeated during the cross-validation process to get the remaining estimate of the model quality.

2. Confusion Matrix:

A **confusion matrix**, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a **matching matrix**). Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see whether the system is confusing two classes (i.e., commonly mislabelling one as another).

It is a special kind of contingency table, with two dimensions ("actual" and "predicted"), and identical sets of "classes" in both dimensions (each combination of dimension and class is a variable in the contingency table).

3. Classification Report:

The classification report visualizer displays the precision, recall, F1, and support scores for the model. There are four ways to check if the predictions are right or wrong:

1. **TN / True Negative**: the case was negative and predicted negative

2. **TP / True Positive**: the case was positive and predicted positive

3. **FN / False Negative**: the case was positive but predicted negative

4. **FP / False Positive**: the case was negative but predicted positive

Precision: 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 a true positive and false positive. It is the accuracy of positive predictions. The formula of precision is given below:

$$\text{Precision} = TP/ (TP + FP)$$

Recall: 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. It is also the fraction of positives that were correctly identified. The formula of recall is given below:

$$\text{Recall} = TP/(TP+FN)$$

F1 score: 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. F1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy. The formula is:

$$\text{F1 Score} = 2*(\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$$

Support: Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

4. Hyperparameter Tuning:

There is a list of different machine learning models. They all are different in some way or the other, but what makes them different is nothing but input parameters for the model. These input parameters are named as **Hyperparameters.** These hyperparameters will define the architecture of the model, and the best part about these is that you get a choice to select these for your model. You must select from a specific list of hyperparameters for a given model as it varies from model to model.

We are not aware of optimal values for hyperparameters which would generate the best model output. So, what we tell the model is to explore and select the optimal model architecture automatically. This selection procedure for hyperparameter is known as **Hyperparameter Tuning. We can do tuning by using GridSearchCV.**

GridSearchCV is a function that comes in Scikit-learn (or SK-learn) model selection package. An important point here to note is that we need to have Scikit-learn library installed on the computer. This function helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, we can select the best parameters from the listed hyperparameters.

Hyperparameter Tuning

```
#Using the best parameters obtained
gbc=GradientBoostingClassifier(random_state=76,n_estimators=100)
gbc.fit(x_train,y_train)
pred=gbc.predict(x_test)
print("Accuracy score: ",accuracy_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(gbc,x,y,cv=3,scoring='accuracy').mean()*100)
print('Classification report: \n')
print(classification_report(y_test,pred))
print('Confusion matrix: \n')
print(confusion_matrix(y_test,pred))
```

```
Accuracy score:  90.42999999999999
Cross validation score:  51.27808556273372
Classification report:

              precision    recall  f1-score   support

         1.0       0.95      0.95      0.95      2295
         2.0       0.98      0.94      0.96       401
         3.0       0.95      0.75      0.84       731
         4.0       0.97      0.68      0.80      1657
         5.0       0.86      0.98      0.92      4916

    accuracy                           0.90     10000
   macro avg       0.94      0.86      0.89     10000
weighted avg       0.91      0.90      0.90     10000

Confusion matrix:

[[2172    4    3    4  112]
 [  11  378    0    1   11]
 [  18    0  549    3  161]
 [  30    2   11 1133  481]
 [  61    2   15   27 4811]]
```

After applying Hyperparameter Tuning, we can see that RandomForestClassifier Algorithm is performing well as the scores are improved, i.e., accuracy score from 91.4 to 91.5 and cross_val_score from 56.344 to 56.346. Now, we will finalize Random ForestClassifier algorithm model as the final model.

FINAL THE MODEL

```
rfc_prediction=rfc.predict(x)

#Making a dataframe of predictions
rating_prediction=pd.DataFrame({'Predictions':rfc_prediction})
rating_prediction
```

	Predictions
0	1.0
1	2.0
2	5.0
3	1.0
4	5.0
...	...
49995	5.0
49996	5.0
49997	5.0
49998	5.0
49999	5.0

50000 rows × 1 columns

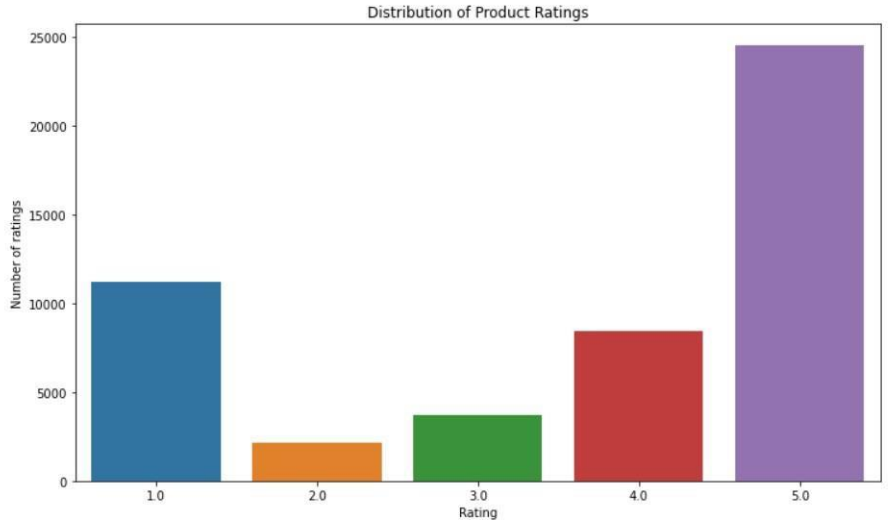
Saving the model

```
#Saving the model
import pickle
filename='RatingsPrediction_Project.pkl'   #Specifying the filename
pickle.dump(rfc,open(filename,'wb'))
```

DATA VISUALIZATION

```
#Importing Matplotlib and Seaborn
import seaborn as sns
import matplotlib.pyplot as plt
```

```
f, axes = plt.subplots(figsize=(12,7))
ax = sns.countplot(x=df['Ratings'])
ax.set(title="Distribution of Product Ratings", xlabel="Rating", ylabel="Number of ratings")
plt.show()
```



CONCLUSION

Key Findings and Conclusions of the Study

-> After the completion of this project, we got an insight of how to collect data, pre-processing the data, analysing the data and building a model.

-> First, we collected the reviews and ratings data from different e-commerce websites like Amazon and Flipkart and it was done by using Webscraping. The framework used for webscraping was Selenium, which has an advantage of automating our process of collecting data.

-> We collected almost 50000 of data which contained the ratings from 1.0 to 5.0 and their reviews.

-> Then, the scrapped data was combined in a single dataframe and saved in a csv file so that we can open it and analyze the data.

-> **We did the pre-processing using NLP and the steps are as follows:**

- Removing Punctuations and other special characters
- Splitting the comments into individual words
- Removing Stop Words
- Stemming and Lemmatising
- Applying Count Vectorizer
- Splitting dataset into Training and Testing

-> After separating our train and test data, we started running different machine learning classification algorithms to find out the best performing model.

-> We found that RandomForest and GradientBoosting Algorithms were performing well, according to their accuracy and cross val scores.

-> Then, we performed Hyperparameter Tuning techniques using GridSearchCV for getting the best parameters and improving the scores. In that, RandomForestClassifier performed well and we finalised that model.

-> We saved the model in pkl format and then saved the predicted values in a csv format.

-> **The problems we faced during this project were:**

- More time consumption during hyperparameter tuning forboth models, as the data was large.
- Less number of parameters were used during tuning.
- Scrapping of data from different websites were of different process and the length of data were differing in most cases.
- Some of the reviews were bad and the text had more wrong information about the product.
- WordCloud was not showing proper text which had more positive and negative weightage.

-> **Areas of improvement:**

- Less time complexity
- More computational power can be given
- Less errors can be avoided.

THANKYOU