International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-10, Issue-6, November 2022
https://doi.org/10.55524/ijircst.2022.10.6.20
Article ID IRPV1075, Pages 120-137
www.ijircst.org

Sentiment Based Product Recommendation System for E-Commerce Using Machine Learning Approaches

Muzakkiruddin Ahmed Mohammed

B. Tech Scholar, Department of Electrical and Electronics Engineering, Lords Institute of Engineering and Technology, Hyderabad, India

ABSTRACT- Today, e-commerce is a thriving industry. We do not need to approach every customer to accept their orders here. A business creates a website to offer things to clients, who can then purchase the stuff they need within the same website. These e-commerce firms include well-known ones like Amazon, Shopify, Myntra, Flipkart, and Ajio. To create a product recommendation system for the end customers, we will be using the data set of e-commerce product reviews in this final project. A sentiment analysis model will be used to enhance the suggestions. Under this final project, we will develop a sentiment analysis engine utilising a variety of machine learning approaches before selecting the model that produces the best results.

KEYWORDS- Recommender Systems; Logistic Regression and Analysis; Random Forest; Xgboost; Hyperparameter Tuning; Deployment.

I. INTRODUCTION

Building an end-to-end recommendation system to suggest things to customers and they'll most likely enjoy based on their purchase history would be fascinating. Customer reviews and ratings for items are extremely important in buying decisions [1]. Assume we wish to purchase a certain product from any e-commerce website. We look at the product reviews, and if individuals provide positive comments or higher ratings, the product is a great buy; if we see more negative comments or lower ratings, the product is not a great investment. This recommendation system, on the other hand, is built on the ability to automate this process and then propose things to other customers based on the already existing data.

Let's say we are a machine learning engineer at an online retailer. This online retailer offers products in a variety of categories, including kitchen and dining items, books, personal care items, pharmaceuticals, cosmetics, and beauty products. It also sells products in a number of other disciplines, including the sale of electrical appliances, home goods, and health care items. Due to the rapid development of technology, online retailers must compete with established industry players like Amazon, Flipkart, and others in order to grow swiftly in the e-commerce sector and establish themselves as key players [2]. We are being tasked as a senior ML engineer with developing a model that would enhance the suggestions sent to users based on their prior reviews and ratings. We intended to do this by developing a sentiment-based product recommendation system, which entails the subsequent activities.

- Researching data and analysing sentiment
- Construction of a recommendation system
- Using the sentiment analysis model to improve the suggestions
- The end-to-end project's user interface deployment

Under this final project, we will develop a sentiment analysis engine utilising a variety of machine learning approaches before selecting the model that produces the best results. The creation of the recommender systems will be the next stage of this project.

II. UNDERSTANDING RECOMMENDER SYSTEMS

How quickly we can find the item we want to buy on an e-commerce platform is one of the key elements that encourages us to actually make the transaction. This may be done, for instance, by using Amazon's "Customers that purchased this item usually bought..." area to generate recommendations.

Systems that use suggestion inputs from users and aggregate them before sending them to the right recipients are known as recommender systems. Further definitions include a system that generates customised suggestions as output or one that directs the user in a tailored manner toward interesting items within a wider range of available choices. In the not-too-distant future, recommender systems will play a crucial role in the media and entertainment sector [4]. Collaborative recommender systems, Content-based recommender systems, Demographic-based recommender systems, Utility-based recommender systems, Knowledge-based recommender systems, and Hybrid recommender systems are the main six types of recommender systems that are used in the Media and Entertainment industry [5].

We witnessed two popular forms of recommendation algorithms: collaborative filtering and content-based filtering [6] . Start with the content-based filtering of the two different recommendation systems. On e-commerce sites, users are frequently given recommendations for particular goods based on previous purchases or the products that users would have previously looked up. These are based on filtering that is content-based. The magnitude difference is now considerably bigger than it was when the item vectors just included logical values. As a result, the recommender system will be better able to distinguish between the products. User-based Collaborative filtering will be covered in the following section.

It's necessary to move on to collaborative filtering, the 2nd most popular form of filtering now that we understand content-based filtering. This is based on the straightforward tenet that a user's preferences are strongly connected with what other users who are similar to them have previously enjoyed. We will discover more about item-based collaborative filtering over the next section.

We will now comprehend about item-based collaborative filtering after learning about user-based collaborative filtering first. User-Based Collaborative Filtering is a method for predicting the products that a user would enjoy using the ratings provided to that product by other customers whose tastes are identical to the targeted users. Collaborative filtering is a popular technique for creating recommendation systems on internet. [8] After getting a sense of the many classifications of recommendation systems, let's examine some real-world issues that frequently arise while developing a recommendation model.

A. Cold Start

This issue occurs whenever new users or new items are added to the system. Since a new item cannot be recommended to users at first once it is added to the recommender system without even any ratings or reviews, it is difficult to predict the preference or curiosity of customers, which results towards less accurate recommendations.

B. Sparsity

It frequently occurs that most customers don't rate or review the products they buy, making the rating model highly sparse and potentially causing data sparsity issues. This makes it harder to identify groups of users that share ratings or preferences.

C. Synonymy

Synonymy occurs when an individual unit is described by two or more separate names or lists of products with similar meanings; in this case, the recommender system is not able to determine if the words depict different products or the identical product.

D. Privacy

In general, a user must feed his individual data (have expertise with hyper-personalization) to the recommender systems in order to receive more valuable services, but this raises concerns about confidentiality of information, data privacy and security. [13] As a result, many users are hesitant to serve their own personal information into recommendation engines some of which have data concerns about privacy. The recommender systems is required to collect and use personal information from users in order to deliver customised suggestion services. To address this challenge, recommender systems need to build confidence among its consumers.

E. Scalability

The scalability of algorithms with real-world datasets under the recommender systems represents a significant challenge. A large amount of changing data is created by customer interaction mostly in form of user ratings, user reviews and hence scalability is indeed a major worry for these datasets.

Recommender system inefficiently interpret findings from huge datasets; certain advanced large-scaled algorithms are necessary to address this issue [7].

F. Latency

We see that numerous items are being added more often to the database of recommender system; however, only previously existing products are recommended to consumers since recently introduced products have not yet been rated. As a result, the issue of latency develops. To address this issue, the collaborative filtering technique and category-based strategy, in conjunction with user-item interaction, can be implemented.

III. PROBLEM STATEMENT

Today, e-commerce is a thriving industry. We do not need to approach every customer to accept their orders here. A business creates a website to offer things to clients, who can then purchase the stuff they need within the same website. These e-commerce firms include well-known ones like Amazon, Shopify, Myntra, Flipkart, and Ajio. Let's say we are a machine learning engineer at an online retailer. This online retailer offers products in a variety of categories, including kitchen and dining items, books, personal care items, pharmaceuticals, cosmetics, and beauty products. It also sells products in a number of other disciplines, including the sale of electrical appliances, home goods, and health care items. Due to the rapid development of technology, online retailers must compete with established industry players like Amazon, Flipkart, and others in order to grow swiftly in the e-commerce sector and establish themselves as key players. We are being tasked as a senior ML engineer with developing a model that would enhance the suggestions sent to users based on their prior reviews and ratings. We intended to do this by developing a sentiment-based product recommendation [9] system, which entails the subsequent activities

- Data acquisition and sentiment analysis
- Setting up a proper recommendation system
- Utilizing sentiment analysis model to improve suggestions
- Deploying a complete project with a user interface
- 30,000 reviews for more than 200 distinct goods make up this dataset (figure 4). There are almost 20,000 individuals who have left ratings and feedback.

The first task's steps are outlined below.

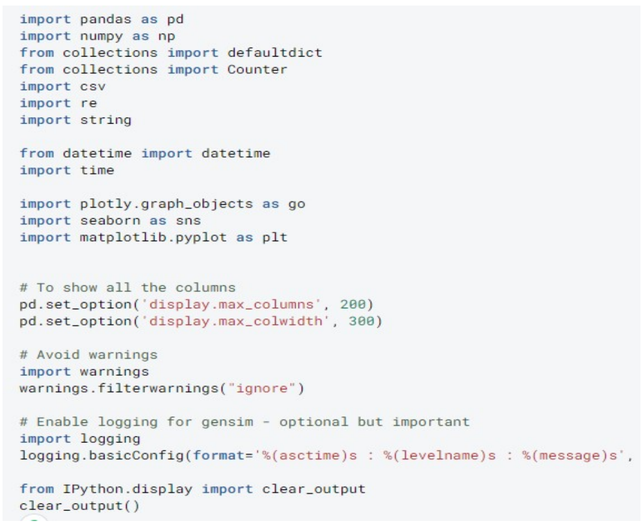
- Analyzing exploratory data (see figure 1-4).
- Cleaning of data (see figure 5-8).
- Text preprocessing (see figure 9-12).
- Feature extraction: To extract features from text data, we can use any of the methods available, such as bag-of-words, TF-IDF vectorization, or word embedding.
- Text classification model training: We must create at least three ML models. We must then evaluate the performance of each of these models and select the best model.
- Logistic regression (see figure 18, figure 21)
- Random forest (see figure 23, figure 26)
- XGBoost (see figure 23, figure 26)
- Naive Bayes (see figure 23)
- Building a recommendation system: We must choose one categorization model from these four depending on its performance. Putting together a recommendation system as previously stated, we can employ the following sorts of recommendation systems.

  - ➢ System of user-generated recommendations
  - ➢ System of item-based recommendations

- Our objective is to examine the recommendation systems and choose the one that is most suited for this situation.
- Using the sentiment analysis model to improve suggestions: The next step is to connect this recommendation system to the sentiment analysis model that was created earlier. When we utilise the recommender system to propose 20 things to a specific user, we must filter out the 5 best products based on the feelings of the 20 recommended product reviews. As a result, we will get an ML model (for feelings) as well as the best-suited recommendation system. Following that, we must make the entire project public.
- Deployment of this whole project, complete with a user interface: We will deploy the entire project once we have obtained the ML model and the best-suited recommendation system. To deploy machine learning models, we must utilise the Flask framework that is commonly used to construct web apps. We must utilise Heroku to make the web application public. Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate apps fully on the internet.

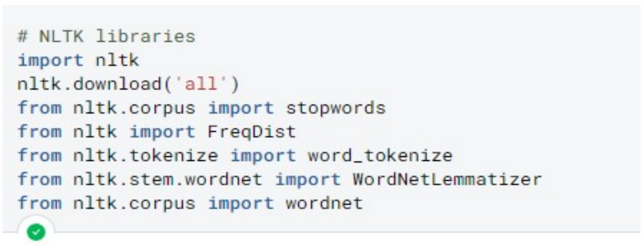
IV. DATA LOADING AND DATA CLEANING

We will import some helpful Python modules and load the data in the form of a data frame in this step. After the data has been put into the data frame, we will go through the data cleaning process to remove any missing values. Now let us begin cleaning up and preparing the data.



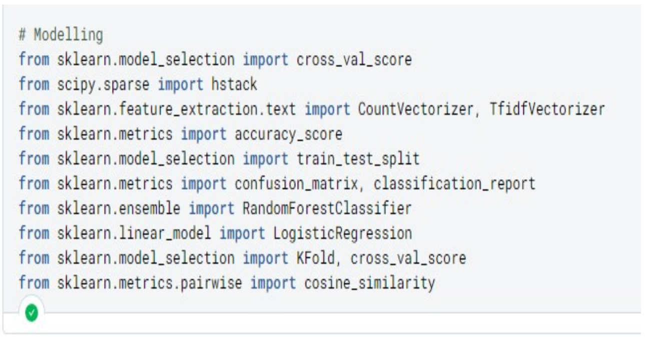
```
import pandas as pd
import numpy as np
from collections import defaultdict
from collections import Counter
import csv
import re
import string

from datetime import datetime
import time

import plotly.graph_objects as go
import seaborn as sns
import matplotlib.pyplot as plt


# To show all the columns
pd.set_option('display.max_columns', 200)
pd.set_option('display.max_colwidth', 300)

# Avoid warnings
import warnings
warnings.filterwarnings("ignore")

# Enable logging for gensim - optional but important
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',

from IPython.display import clear_output
clear_output()
```

Figure 1: Import Packages



```
# NLTK libraries
import nltk
nltk.download('all')
from nltk.corpus import stopwords
from nltk import FreqDist
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
```

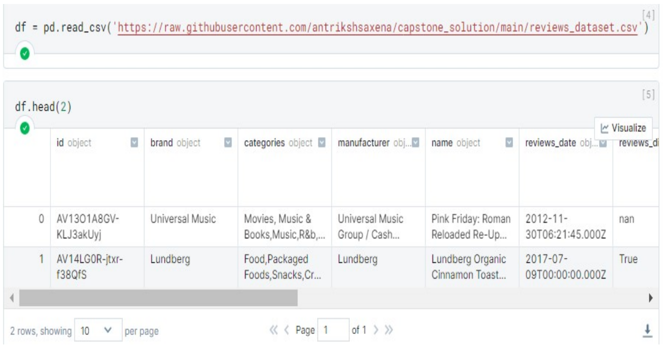
Figure 2: NLTK libraries



```
# Modelling
from sklearn.model_selection import cross_val_score
from scipy.sparse import hstack
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics.pairwise import cosine_similarity
```

Figure 3: Modelling

In this, we first loaded the data after importing a few helpful Python modules. We loaded the data set from a GitHub repository. Additionally, we discovered some missing values in the data and deleted them. In the following, we performed fundamental data cleaning to eliminate missing values from a data collection.



```
df = pd.read_csv('https://raw.githubusercontent.com/antrikshsaxena/capstone_solution/main/reviews_dataset.csv')
```

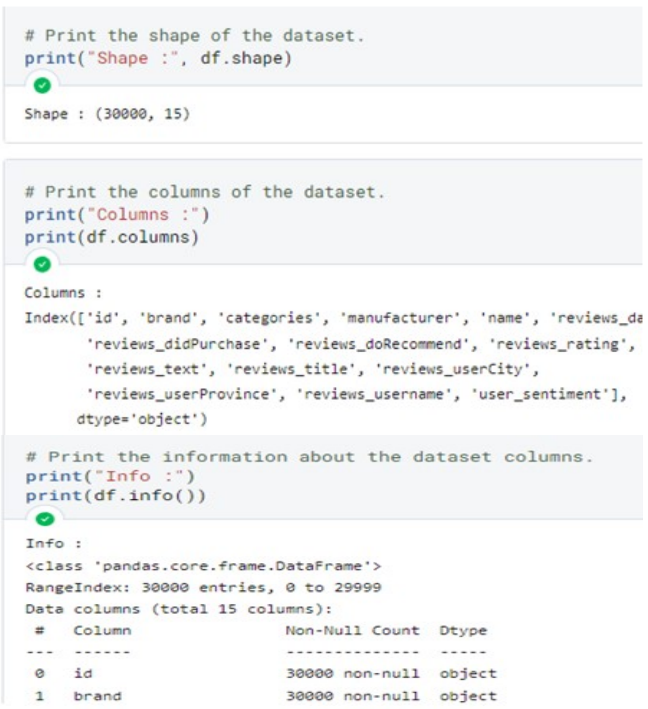
```
df.head(2)
```

	id object	brand object	categories object	manufacturer obj...	name object	reviews_date obj...	reviews_di
0	AV13O1A8GV-KLJ3akUyj	Universal Music	Movies, Music & Books,Music,R&b,...	Universal Music Group / Cash...	Pink Friday: Roman Reloaded Re-Up...	2012-11-30T06:21:45.000Z	nan
1	AV14LGOR-jtxr-f38QfS	Lundberg	Food,Packaged Foods,Snacks,Cr...	Lundberg	Lundberg Organic Cinnamon Toast...	2017-07-09T00:00:00.000Z	True

2 rows, showing 10 per page   Page 1 of 1

Figure 4: Load the dataset

In the following, we performed fundamental data cleaning (figure 5) to eliminate missing values from a data collection.



```
# Print the shape of the dataset.
print("Shape :", df.shape)
```

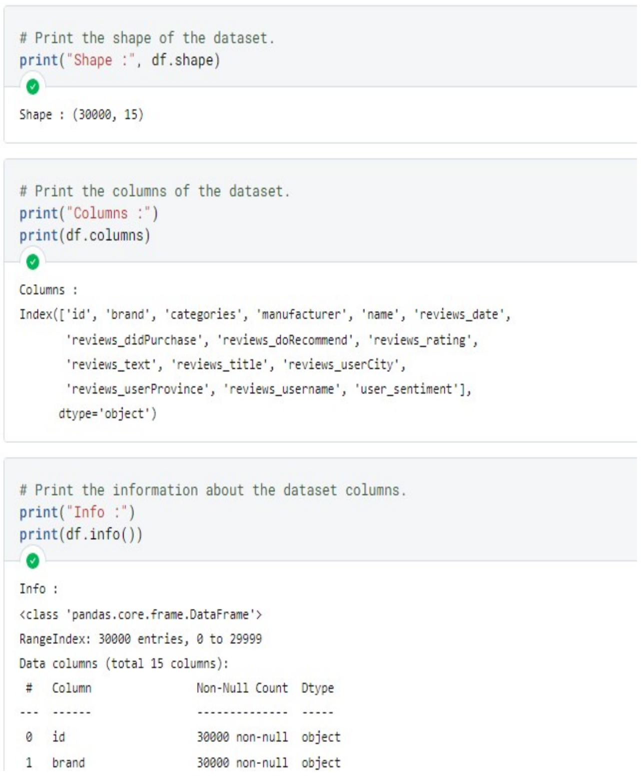
```
Shape : (30000, 15)
```

```
# Print the columns of the dataset.
print("Columns :")
print(df.columns)
```

```
Columns :
Index(['id', 'brand', 'categories', 'manufacturer', 'name', 'reviews_da
       'reviews_didPurchase', 'reviews_doRecommend', 'reviews_rating',
       'reviews_text', 'reviews_title', 'reviews_userCity',
       'reviews_userProvince', 'reviews_username', 'user_sentiment'],
      dtype='object')
```

```
# Print the information about the dataset columns.
print("Info :")
print(df.info())
```

```
Info :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 15 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    30000 non-null  object
 1   brand                 30000 non-null  object
```



```
# Print the shape of the dataset.
print("Shape :", df.shape)
```

```
Shape : (30000, 15)
```

```
# Print the columns of the dataset.
print("Columns :")
print(df.columns)
```

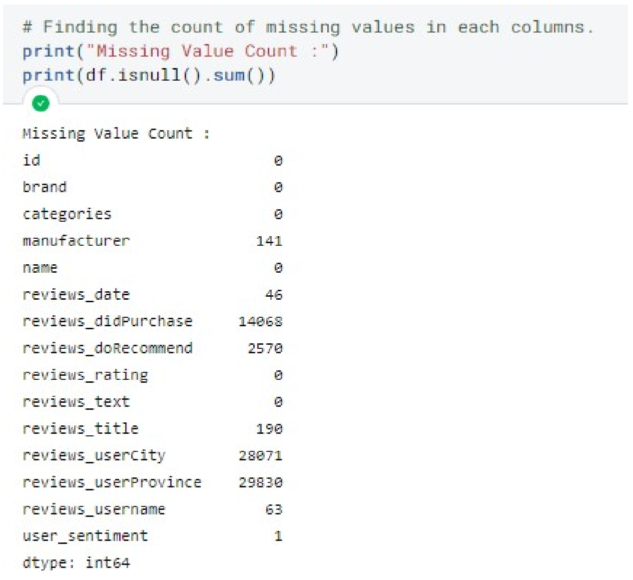
```
Columns :
Index(['id', 'brand', 'categories', 'manufacturer', 'name', 'reviews_date',
       'reviews_didPurchase', 'reviews_doRecommend', 'reviews_rating',
       'reviews_text', 'reviews_title', 'reviews_userCity',
       'reviews_userProvince', 'reviews_username', 'user_sentiment'],
      dtype='object')
```

```
# Print the information about the dataset columns.
print("Info :")
print(df.info())
```

```
Info :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 15 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    30000 non-null  object
 1   brand                 30000 non-null  object
```

Figure 5: Data cleaning

The columns'reviews didPurchase,''reviews userCity,' and'reviews userProvince' have a very high proportion of missing data in this section. To address this, the data frame's columns with more than 50% null values was eliminated. The 'reviews userCity' and'reviews userProvince' columns, as well as 4 the'reviews doRecommend' and 'reviews didPurchase' columns, were also removed since they were irrelevant to our analyses.

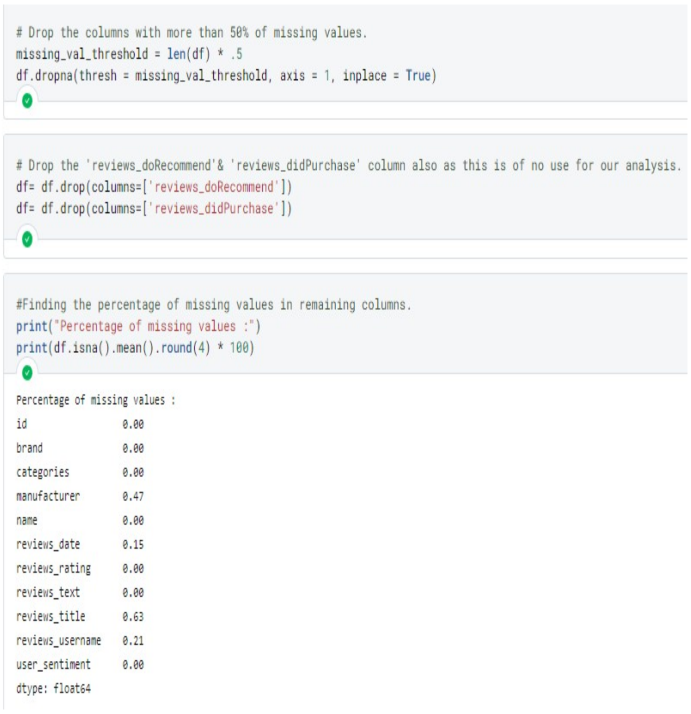


```
# Finding the count of missing values in each columns.
print("Missing Value Count :")
print(df.isnull().sum())
```

```
Missing Value Count :
id                          0
brand                       0
categories                  0
manufacturer              141
name                        0
reviews_date               46
reviews_didPurchase     14068
reviews_doRecommend      2570
reviews_rating              0
reviews_text                0
reviews_title             190
reviews_userCity        28071
reviews_userProvince    29830
reviews_username           63
user_sentiment              1
dtype: int64
```

Figure 6: Finding the percentage of missing values in remaining columns

We also eliminated rows with null values in columns like'reviews text,''reviews title,''reviews username,' 'user sentiment,''reviews date,' and'manufacture.'



```
# Drop the columns with more than 50% of missing values.
missing_val_threshold = len(df) * .5
df.dropna(thresh = missing_val_threshold, axis = 1, inplace = True)
```

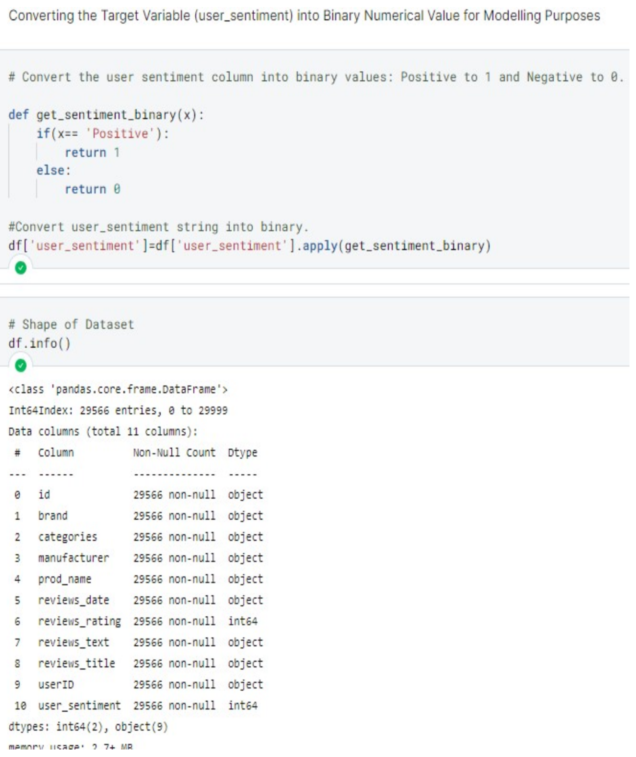
```
# Drop the 'reviews_doRecommend'& 'reviews_didPurchase' column also as this is of no use for our analysis.
df= df.drop(columns=['reviews_doRecommend'])
df= df.drop(columns=['reviews_didPurchase'])
```

```
#Finding the percentage of missing values in remaining columns.
print("Percentage of missing values :")
print(df.isna().mean().round(4) * 100)
```

```
Percentage of missing values :
id                  0.00
brand               0.00
categories          0.00
manufacturer        0.47
name                0.00
reviews_date        0.15
reviews_rating      0.00
reviews_text        0.00
reviews_title       0.63
reviews_username    0.21
user_sentiment      0.00
dtype: float64
```

Figure 7: Dropping the columns with more than 50% missing value

As we can see from the data frame, the value in the 'user sentiment' column is either 'Positive' or 'Negative,' which cannot be utilised as a target variable for a classification model because it is a string data type. To address this issue, values were transformed to binary using the user-created function 'get sentiment binary(x),' which transforms positive sentiment to 1 and negative sentiment to 0.



Converting the Target Variable (user_sentiment) into Binary Numerical Value for Modelling Purposes

```
# Convert the user sentiment column into binary values: Positive to 1 and Negative to 0.

def get_sentiment_binary(x):
    if(x== 'Positive'):
        return 1
    else:
        return 0

#Convert user_sentiment string into binary.
df['user_sentiment']=df['user_sentiment'].apply(get_sentiment_binary)
```

```
# Shape of Dataset
df.info()
```

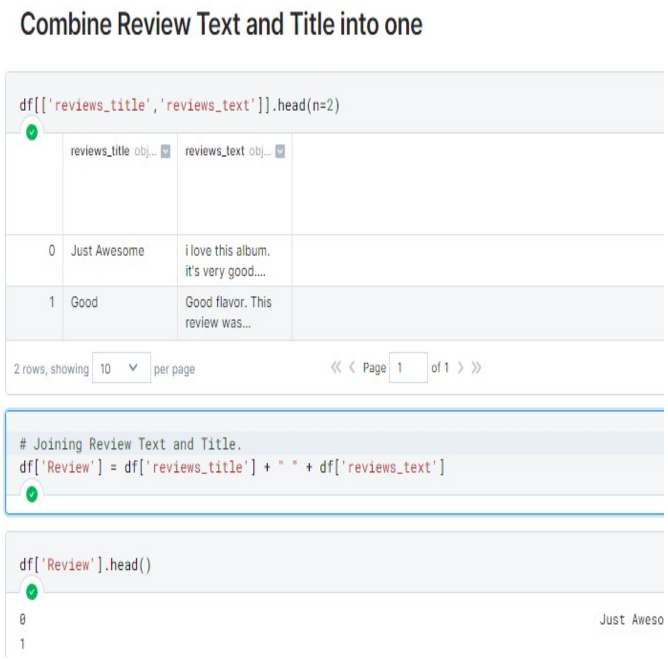
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29566 entries, 0 to 29999
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              29566 non-null  object
 1   brand           29566 non-null  object
 2   categories      29566 non-null  object
 3   manufacturer    29566 non-null  object
 4   prod_name       29566 non-null  object
 5   reviews_date    29566 non-null  object
 6   reviews_rating  29566 non-null  int64
 7   reviews_text    29566 non-null  object
 8   reviews_title   29566 non-null  object
 9   userID          29566 non-null  object
 10  user_sentiment  29566 non-null  int64
dtypes: int64(2), object(9)
memory usage: 2.7+ MB
```

Figure 8: Converting the target variable into binary numerical value

The next section will demonstrate able to successfully cleanup textual data employing text processing technologies.

V. TEXT PROCESSING

We loaded the data in the preceding section and reduced the number of missing values to nearly 0%. In this section, we will execute preprocessing processes on the textual data to further clean it. We used the fundamental text preparation techniques described below in this section:
Combining the review text and title into one column: Since the columns "reviews title" and "reviews text" both include a textual representation of the emotion, combining them into one column, "Review," would be helpful for the analysis.


Combine Review Text and Title into one

```
df[['reviews_title','reviews_text']].head(n=2)
```

	reviews_title	reviews_text
0	Just Awesome	i love this album. it's very good....
1	Good	Good flavor. This review was...

2 rows, showing 10 per page   Page 1 of 1

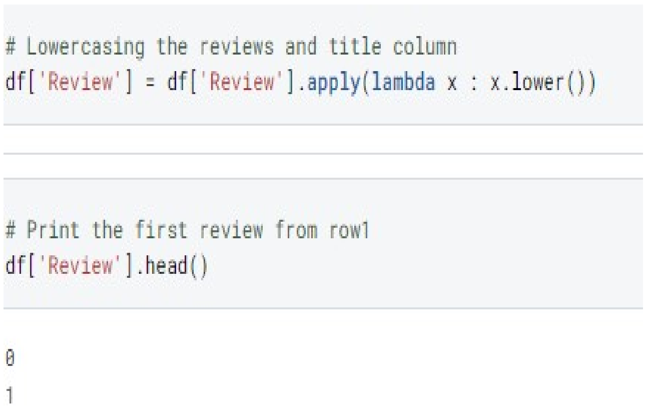
```
# Joining Review Text and Title.
df['Review'] = df['reviews_title'] + " " + df['reviews_text']
```

```
df['Review'].head()
```

```
0                                                Just Aweso
1
```

Figure 9: Combine Review Text and Title into one

Lowercasing: Lowercasing is carried out as a best practise and to steer clear of any problems brought on by the text's case sensitivity.



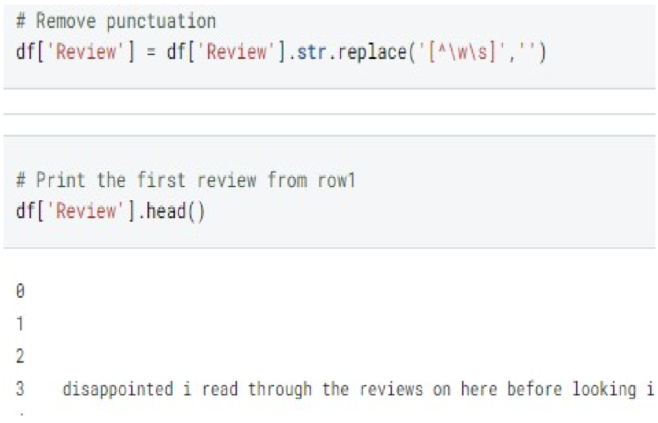
```
# Lowercasing the reviews and title column
df['Review'] = df['Review'].apply(lambda x : x.lower())
```

```
# Print the first review from row1
df['Review'].head()
```

```
0
1
```

Figure 10: Lowering

Removing punctuation: deleting the punctuation Punctuation should be removed since it serves no use to do so while attempting to extract meaning from text data.



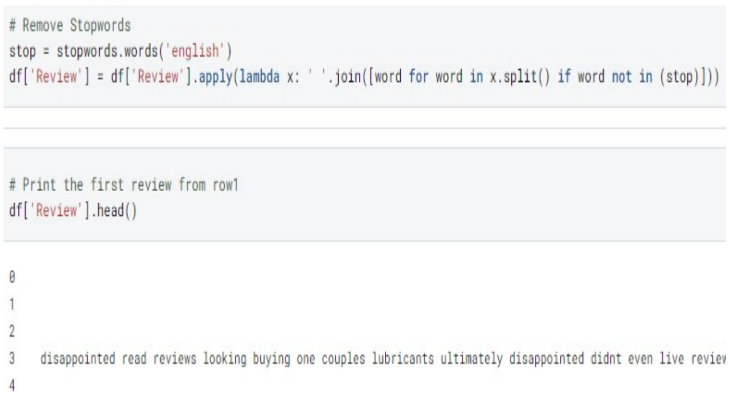
```
# Remove punctuation
df['Review'] = df['Review'].str.replace('[^\w\s]','')
```

```
# Print the first review from row1
df['Review'].head()
```

```
0
1
2
3    disappointed i read through the reviews on here before looking i
```

Figure 11: Removing punctuation

Eliminating stop words: Stopwords like "the," "in," "on," and "is" contain little to no meaning when meaning is being extrapolated from text. So, getting rid of them is a smart idea(see figure 12, and 15).



```
# Remove Stopwords
stop = stopwords.words('english')
df['Review'] = df['Review'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
```

```
# Print the first review from row1
df['Review'].head()
```

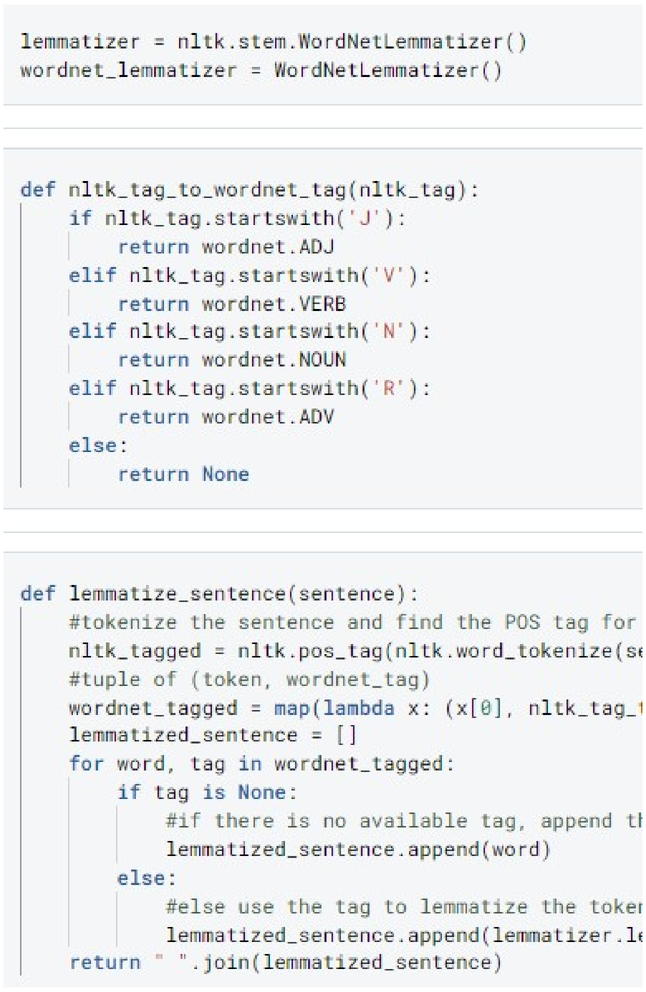
```
0
1
2
3   disappointed read reviews looking buying one couples lubricants ultimately disappointed didnt even live reviev
4
```

Figure 12: Remove Stopwords

VI. LEMMATIZATION

We will use sophisticated text processing techniques like "lemmatization" and "noise reduction" in the next section to further cleanup the text data [3].
Lemmatization is a text processing technique that changes a given word to its lemma (root) form [3], like we saw in the above. For instance, changing "better" and "best" to "good," "learning" to "learn," etc.
Let's now review the lemmatization procedure:

- WordnetLemmatizer lemmatizes the input word and its related wordnet POS tag before joining the two inputs to create a sentence.(see figure 13, and 14)
- By utilising the function "nltk tag to wordnet tag()," we must define the function that converts "nltk POS tag" to "wordnet POS tag" in order to obtain wordnet POS tag.
- We then define the function lemmatize sentence(sentence), which carries out the subsequent actions in the following order:
- Tokenize the phrase and use the function nltk.pos tag(nltk.word tokenize(sentence)) to determine the POS tag of each token (word).



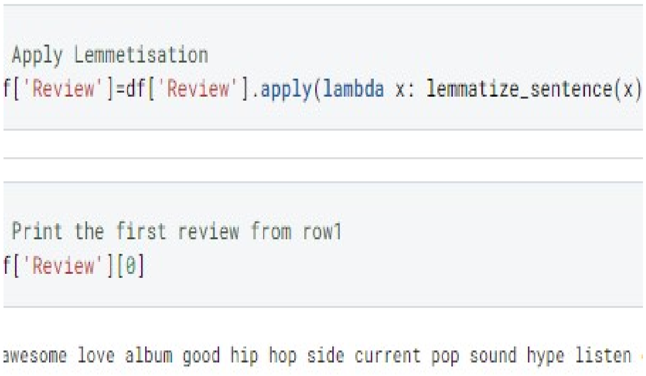
```
lemmatizer = nltk.stem.WordNetLemmatizer()
wordnet_lemmatizer = WordNetLemmatizer()
```

```
def nltk_tag_to_wordnet_tag(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None
```

```
def lemmatize_sentence(sentence):
    #tokenize the sentence and find the POS tag for
    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(se
    #tuple of (token, wordnet_tag)
    wordnet_tagged = map(lambda x: (x[0], nltk_tag_t
    lemmatized_sentence = []
    for word, tag in wordnet_tagged:
        if tag is None:
            #if there is no available tag, append tl
            lemmatized_sentence.append(word)
        else:
            #else use the tag to lemmatize the toker
            lemmatized_sentence.append(lemmatizer.le
    return " ".join(lemmatized_sentence)
```

Figure 13: Lemmatization

- Make a tuple containing the word (token) and the wordnet POS tag that goes with it (which may be produced by the aforementioned function, "nltk tag to wordnet tag").
- After iterating the lemmatizer over the wordnet tagged tuple (using the code "lemmatized sentence.append(lemmatizer.lemmatize(word, tag))"), create a list called "lemmatized sentence" that includes the lemmatized sentence and link them.
- Applying the "lemmatize sentence(sentence)" function on the "Review" column next will remove noise from the text data.



```
 Apply Lemmetisation
f['Review']=df['Review'].apply(lambda x: lemmatize_sentence(x)
```

```
 Print the first review from row1
f['Review'][0]
```

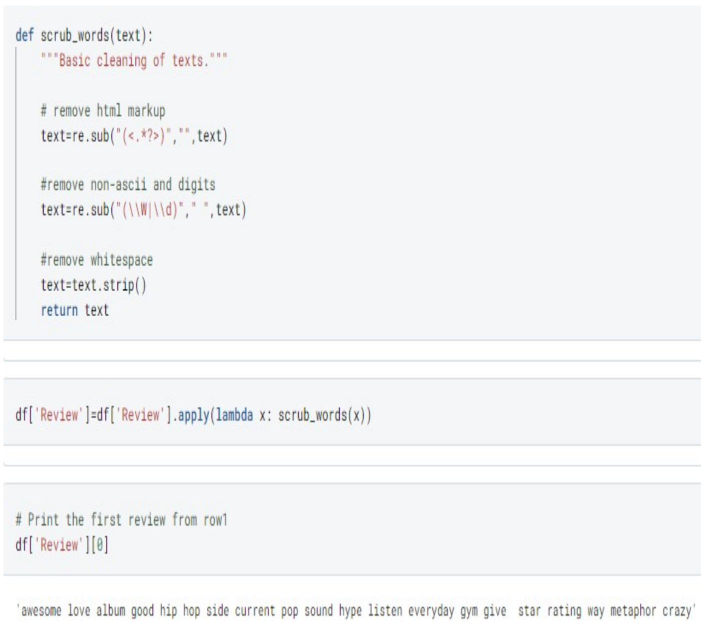
```
awesome love album good hip hop side current pop sound hype listen
```

Figure 14: Lemmatization

VII. NOISE REDUCTION APPROACH

We used the noise reduction approach in this project, which entails the following steps (see figure 15):

- Deleting HTML tags
- Elimination of white space
- Removal of numbers and characters that aren't ASCII
- We have so far used a variety of text processing techniques to clean the data and improve our findings.
- The reviews text will be used to construct features in the following section utilising vectorizer techniques like TF-IDF.



```
def scrub_words(text):
    """Basic cleaning of texts."""

    # remove html markup
    text=re.sub("(<.*?>)","",text)

    #remove non-ascii and digits
    text=re.sub("(\\W|\\d)"," ",text)

    #remove whitespace
    text=text.strip()
    return text
```

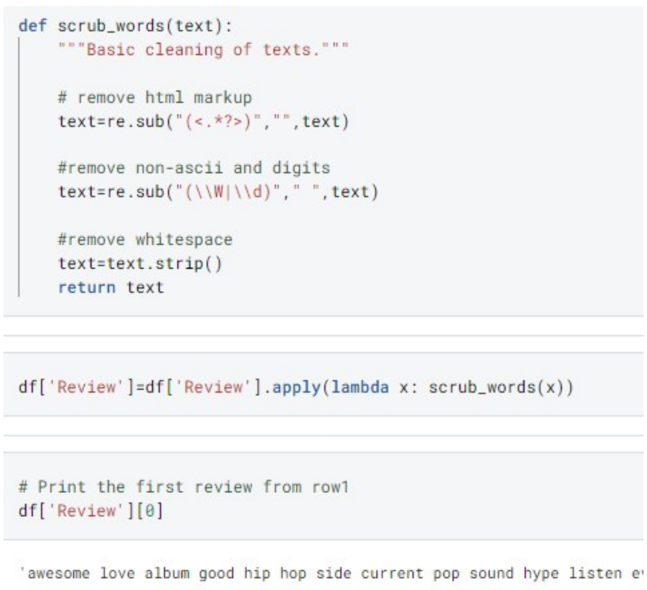
```
df['Review']=df['Review'].apply(lambda x: scrub_words(x))
```

```
# Print the first review from row1
df['Review'][0]
```

```
'awesome love album good hip hop side current pop sound hype listen everyday gym give  star rating way metaphor crazy'
```

Figure 15: Remove Stopwords

VIII. TRAIN TEST SPLIT AND TF-IDF VECTORIZER



```
def scrub_words(text):
    """Basic cleaning of texts."""

    # remove html markup
    text=re.sub("(<.*?>)","",text)

    #remove non-ascii and digits
    text=re.sub("(\\W|\\d)"," ",text)

    #remove whitespace
    text=text.strip()
    return text
```

```
df['Review']=df['Review'].apply(lambda x: scrub_words(x))
```

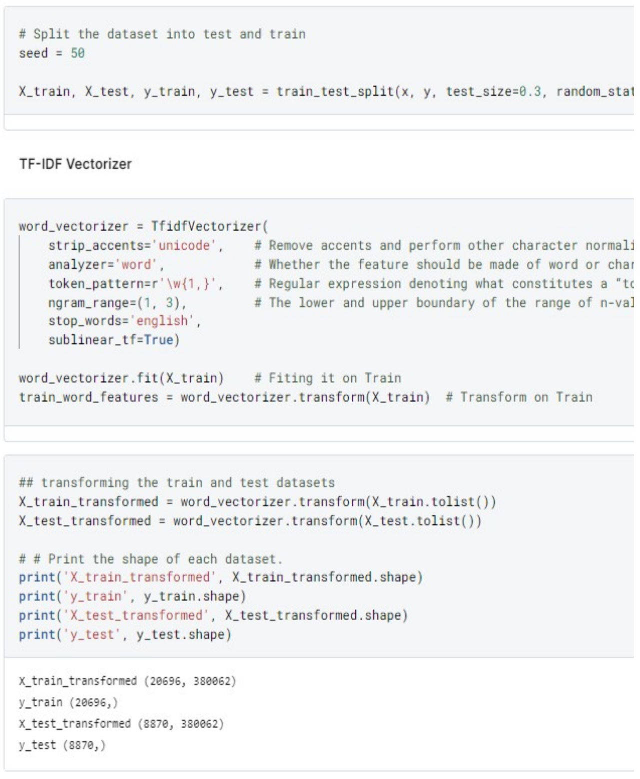
```
# Print the first review from row1
df['Review'][0]
```

```
'awesome love album good hip hop side current pop sound hype listen e
```

Figure 16: Defining features and target variables

We used several text processing techniques to clean up the data set in the previous section.

This text data will now be used for ML modelling [10]. To do this, first divide the data into test and train sets, then describe the features and the target variables, and last construct a TF-IDF vectorizer. Comprehend how to carry out these actions by examining the project.



```
# Split the dataset into test and train
seed = 50

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_stat
```

TF-IDF Vectorizer

```
word_vectorizer = TfidfVectorizer(
    strip_accents='unicode',     # Remove accents and perform other character normali
    analyzer='word',             # Whether the feature should be made of word or char
    token_pattern=r'\w{1,}',     # Regular expression denoting what constitutes a "tc
    ngram_range=(1, 3),          # The lower and upper boundary of the range of n-val
    stop_words='english',
    sublinear_tf=True)

word_vectorizer.fit(X_train)    # Fiting it on Train
train_word_features = word_vectorizer.transform(X_train)  # Transform on Train
```

```
## transforming the train and test datasets
X_train_transformed = word_vectorizer.transform(X_train.tolist())
X_test_transformed = word_vectorizer.transform(X_test.tolist())

# # Print the shape of each dataset.
print('X_train_transformed', X_train_transformed.shape)
print('y_train', y_train.shape)
print('X_test_transformed', X_test_transformed.shape)
print('y_test', y_test.shape)
```

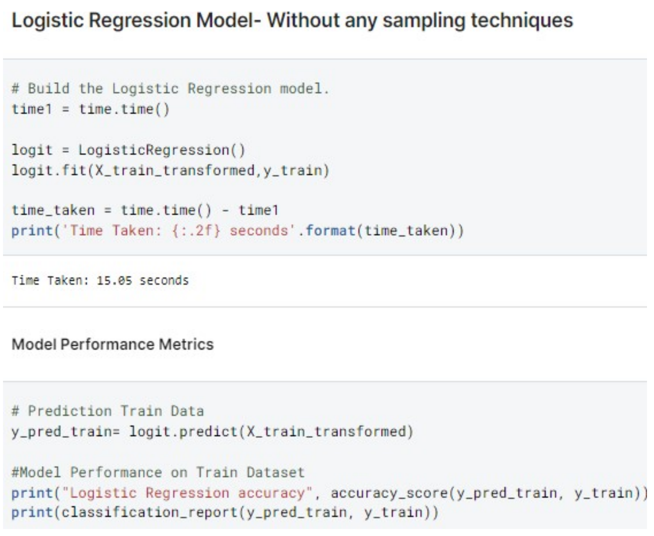
```
X_train_transformed (20696, 380062)
y_train (20696,)
X_test_transformed (8870, 380062)
y_test (8870,)
```

Figure 17: Train Test Split and TF-IDF VECTORIZER

The lessons we observed in this session are summarised as follows:
Only 11.17 percent of the target variable's distribution was negative, with 88.8 percent of responses being positive. Because there was more information regarding positive than negative attitudes, the data was obviously unbalanced.
The data set was then divided into a 70% train data set and 30% test data set. The ML classifier could not be directly fed the data in the train set since it was in text form. It was necessary to transform the data into numerical form. To achieve this, we created a matrix with the TF- IDF score of the review that corresponded to each word as a value in each column, the reviews as rows, and the words from the corpus as columns in the matrix(see figure 16, and 17).
We might also use alternative vectorization techniques besides TF-IDF, including the bag-of-words model or normalised term frequency. To forecast the sentiments, we will use the logistic regression model in the next section.

IX. LOGISTIC REGRESSION AND ANALYSIS

We have thus far cleaned data, developed a TF-IDF vectorizer, and extracted features from the review's text. Now let's use the train data set to create a logistic regression model.
We will construct the logistic regression model without utilising the sampling strategy in the next section of code, and we'll analyse the model's results using several performance metrics.



Logistic Regression Model- Without any sampling techniques

```
# Build the Logistic Regression model.
time1 = time.time()

logit = LogisticRegression()
logit.fit(X_train_transformed,y_train)

time_taken = time.time() - time1
print('Time Taken: {:.2f} seconds'.format(time_taken))
```

```
Time Taken: 15.05 seconds
```

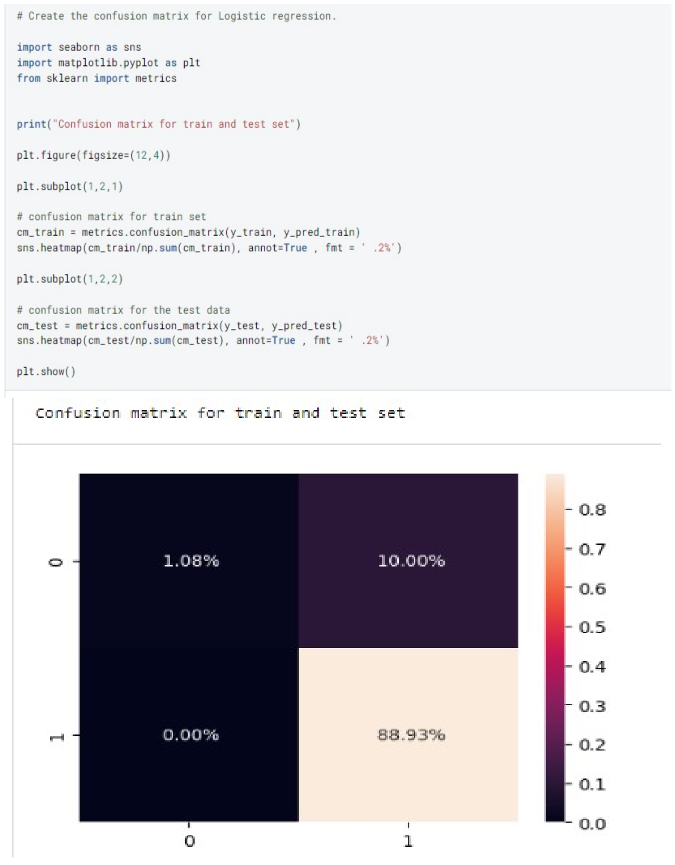
Model Performance Metrics

```
# Prediction Train Data
y_pred_train= logit.predict(X_train_transformed)

#Model Performance on Train Dataset
print("Logistic Regression accuracy", accuracy_score(y_pred_train, y_train))
print(classification_report(y_pred_train, y_train))
```

Figure 18: Logistic Regression and Analysis

In this section, we created a logistic regression model based on the TF-IDF scores assigned to each word in the data set. We also tested the model's performance using two matrices, sensitivity and specificity, for the train and test sets(see figure 21, 24, and 25), respectively.
The fraction of actual positive instances that were projected as positive is referred to as sensitivity.

- True positives / (True positives + False negatives) = Sensitivity
- Specificity is defined as the fraction of genuine negatives that were projected to be negatives.
- True negatives / (False positives + True negatives) = Specificity

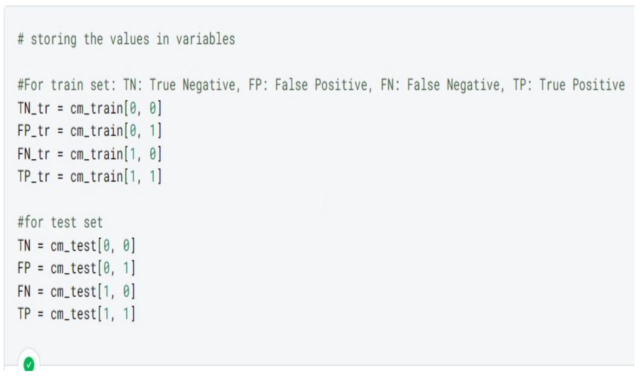


```
# Create the confusion matrix for Logistic regression.

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics


print("Confusion matrix for train and test set")

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)

# confusion matrix for train set
cm_train = metrics.confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm_train/np.sum(cm_train), annot=True , fmt = ' .2%')

plt.subplot(1,2,2)

# confusion matrix for the test data
cm_test = metrics.confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm_test/np.sum(cm_test), annot=True , fmt = ' .2%')

plt.show()
```

```
Confusion matrix for train and test set
```

Figure 19: Confusion Matrix for logistic Regression

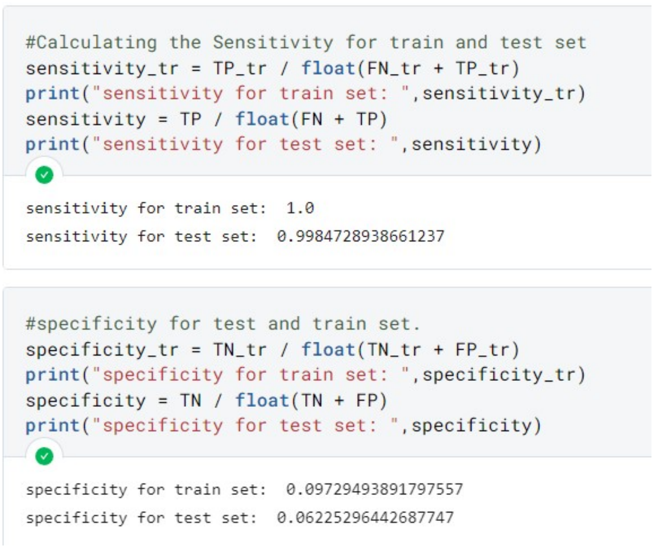
The logistic regression model's specificity without sampling is 9.7% for the train set and 6.2% for the test set, which is not promising for predicting actual negative outcomes. This is because the data set is unbalanced and contains more than 80% favourable evaluations. We shall utilise sampling approaches to address this.



```
# storing the values in variables

#For train set: TN: True Negative, FP: False Positive, FN: False Negative, TP: True Positive
TN_tr = cm_train[0, 0]
FP_tr = cm_train[0, 1]
FN_tr = cm_train[1, 0]
TP_tr = cm_train[1, 1]

#for test set
TN = cm_test[0, 0]
FP = cm_test[0, 1]
FN = cm_test[1, 0]
TP = cm_test[1, 1]
```

Figure 20: Storing the values in variables

In the next section, we will use an oversampling approach to create a logistic regression model.
In this part, we used an oversampling strategy to obtain equal value counts of the target variable, i.e., 1s: 18404 and 0s:18404, which balances the data.



```
#Calculating the Sensitivity for train and test set
sensitivity_tr = TP_tr / float(FN_tr + TP_tr)
print("sensitivity for train set: ",sensitivity_tr)
sensitivity = TP / float(FN + TP)
print("sensitivity for test set: ",sensitivity)
```

```
sensitivity for train set:  1.0
sensitivity for test set:  0.9984728938661237
```

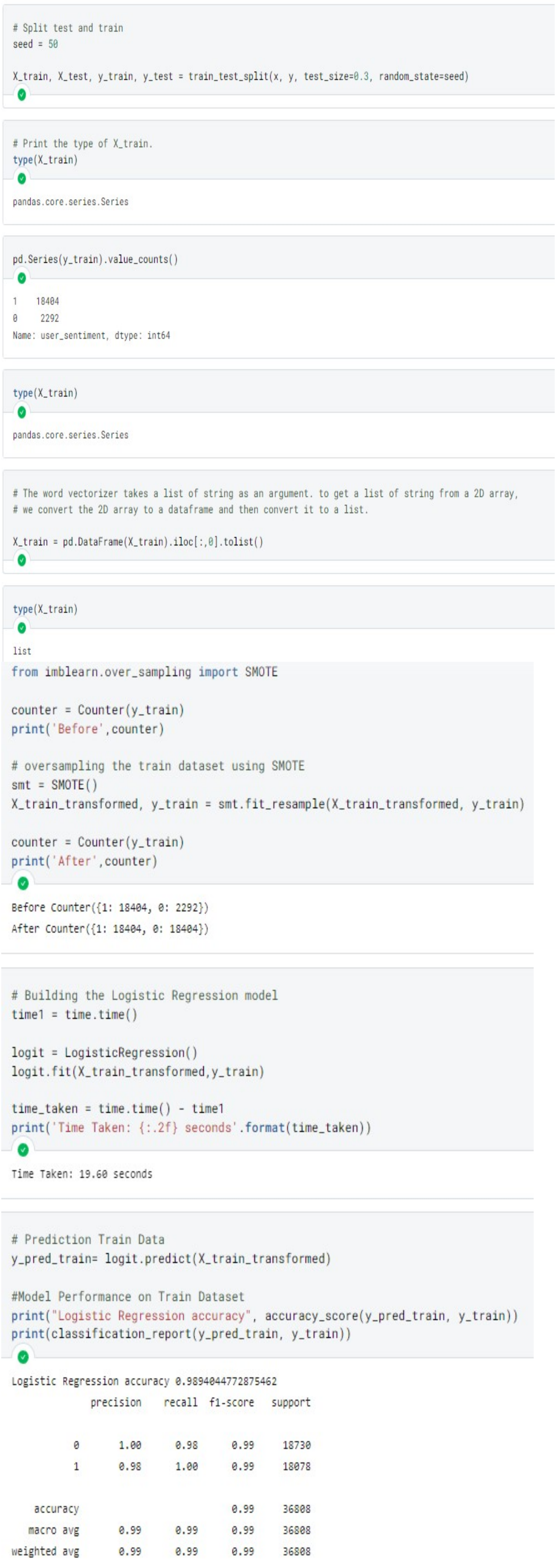
```
#specificity for test and train set.
specificity_tr = TN_tr / float(TN_tr + FP_tr)
print("specificity for train set: ",specificity_tr)
specificity = TN / float(TN + FP)
print("specificity for test set: ",specificity)
```

```
specificity for train set:  0.09729493891797557
specificity for test set:  0.06225296442687747
```

Figure 21: Sensitivity for train and test set

Furthermore, the specificity increased from 9.7% to 99.8% for the train set and from 6.2% to 53.1% for the test set.
In the following session, we will see how to use another sampling strategy called 'Smote,' as well as analyse the performance metrics of the train and test sets.[11]
We used Smote on the train set in this research and followed the same method as in the prior oversampling technique. As a consequence, the model generated with the Smote approach had significantly higher sensitivity than the model built without a sampling technique. However, it improved less in specificity. We will describe all of the approaches utilised in the logistic regression model in the code supplied above in the following section(see figure 18, and 21).

X. LOGISTIC REGRESSION MODEL: SMOTE



```
# Split test and train
seed = 50

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=seed)
```

```
# Print the type of X_train.
type(X_train)
```

```
pandas.core.series.Series
```

```
pd.Series(y_train).value_counts()
```

```
1    18404
0     2292
Name: user_sentiment, dtype: int64
```

```
type(X_train)
```

```
pandas.core.series.Series
```

```
# The word vectorizer takes a list of string as an argument. to get a list of string from a 2D array,
# we convert the 2D array to a dataframe and then convert it to a list.

X_train = pd.DataFrame(X_train).iloc[:,0].tolist()
```

```
type(X_train)
```

```
list
```

```
from imblearn.over_sampling import SMOTE

counter = Counter(y_train)
print('Before',counter)

# oversampling the train dataset using SMOTE
smt = SMOTE()
X_train_transformed, y_train = smt.fit_resample(X_train_transformed, y_train)

counter = Counter(y_train)
print('After',counter)
```

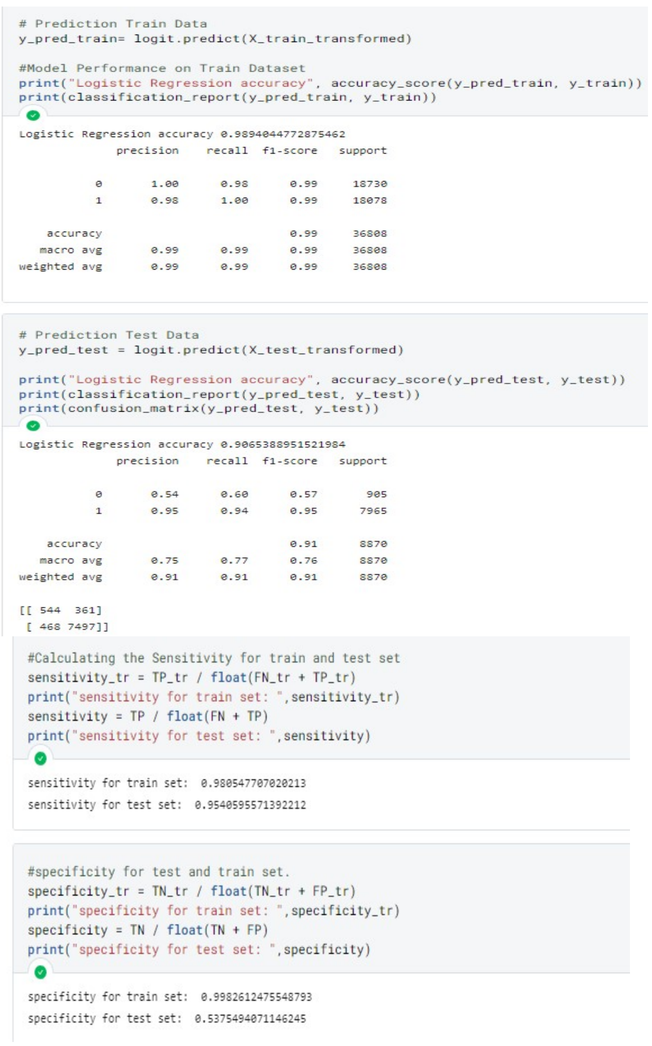
```
Before Counter({1: 18404, 0: 2292})
After Counter({1: 18404, 0: 18404})
```

```
# Building the Logistic Regression model
time1 = time.time()

logit = LogisticRegression()
logit.fit(X_train_transformed,y_train)

time_taken = time.time() - time1
print('Time Taken: {:.2f} seconds'.format(time_taken))
```

```
Time Taken: 19.60 seconds
```

```
# Prediction Train Data
y_pred_train= logit.predict(X_train_transformed)

#Model Performance on Train Dataset
print("Logistic Regression accuracy", accuracy_score(y_pred_train, y_train))
print(classification_report(y_pred_train, y_train))
```

```
Logistic Regression accuracy 0.9894044772875462
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     18730
           1       0.98      1.00      0.99     18078

    accuracy                           0.99     36808
   macro avg       0.99      0.99      0.99     36808
weighted avg       0.99      0.99      0.99     36808
```



```
# Prediction Train Data
y_pred_train= logit.predict(X_train_transformed)

#Model Performance on Train Dataset
print("Logistic Regression accuracy", accuracy_score(y_pred_train, y_train))
print(classification_report(y_pred_train, y_train))
```

```
Logistic Regression accuracy 0.9894044772875462
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     18730
           1       0.98      1.00      0.99     18078

    accuracy                           0.99     36808
   macro avg       0.99      0.99      0.99     36808
weighted avg       0.99      0.99      0.99     36808
```

```
# Prediction Test Data
y_pred_test = logit.predict(X_test_transformed)

print("Logistic Regression accuracy", accuracy_score(y_pred_test, y_test))
print(classification_report(y_pred_test, y_test))
print(confusion_matrix(y_pred_test, y_test))
```

```
Logistic Regression accuracy 0.9065388951521984
              precision    recall  f1-score   support

           0       0.54      0.60      0.57       905
           1       0.95      0.94      0.95      7965

    accuracy                           0.91      8870
   macro avg       0.75      0.77      0.76      8870
weighted avg       0.91      0.91      0.91      8870

[[ 544  361]
 [ 468 7497]]
```

```
#Calculating the Sensitivity for train and test set
sensitivity_tr = TP_tr / float(FN_tr + TP_tr)
print("sensitivity for train set: ",sensitivity_tr)
sensitivity = TP / float(FN + TP)
print("sensitivity for test set: ",sensitivity)
```

```
sensitivity for train set:  0.980547707020213
sensitivity for test set:  0.9540595571392212
```

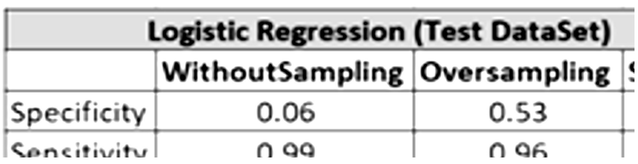
```
#specificity for test and train set.
specificity_tr = TN_tr / float(TN_tr + FP_tr)
print("specificity for train set: ",specificity_tr)
specificity = TN / float(TN + FP)
print("specificity for test set: ",specificity)
```

```
specificity for train set:  0.9982612475548793
specificity for test set:  0.5375494071146245
```

Figure 22: Logistic Regression Model: Smote

As a result, we get the summary report shown above the figure 22.

Table 1: Logistic Regression (Test Dataset)

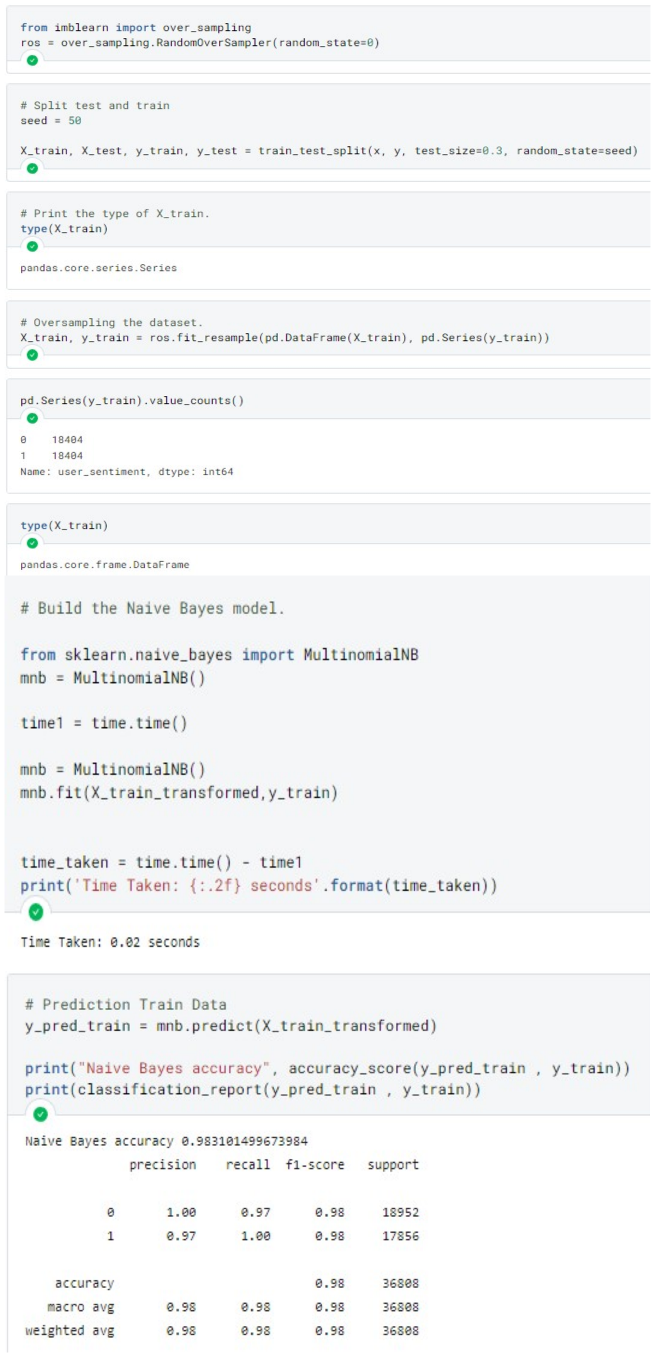


Logistic Regression (Test DataSet)		
	WithoutSampling	Oversampling
Specificity	0.06	0.53
Sensitivity	0.99	0.96

Oversampling also has a higher sensitivity over Smote [12]. As a result, we may decide whether to develop further ML models using oversampling approaches in order to obtain better predictions. In the following part, we will apply additional classification Machine learning Models to the oversampled data.

XI. NAIVE BAYES, RANDOM FOREST AND XGBOOST

We understood about the 'Smote' and 'oversampling' approaches in the previous segment and developed a logistic regression model. Specificity and sensitivity were then utilised also as performance metrics. Oversampling is indeed a promising strategy for dealing with unbalanced data, and it will therefore be utilised for various classification models such as Naive Bayes, Random Forest, and XGBoost(see figure 23, 24 and 25).



```
from imblearn import over_sampling
ros = over_sampling.RandomOverSampler(random_state=0)
```

```
# Split test and train
seed = 50

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=seed)
```

```
# Print the type of X_train.
type(X_train)
```

```
pandas.core.series.Series
```

```
# Oversampling the dataset.
X_train, y_train = ros.fit_resample(pd.DataFrame(X_train), pd.Series(y_train))
```

```
pd.Series(y_train).value_counts()
```

```
0    18404
1    18404
Name: user_sentiment, dtype: int64
```

```
type(X_train)
```

```
pandas.core.frame.DataFrame
```

```
# Build the Naive Bayes model.

from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB()

time1 = time.time()

mnb = MultinomialNB()
mnb.fit(X_train_transformed,y_train)


time_taken = time.time() - time1
print('Time Taken: {:.2f} seconds'.format(time_taken))
```

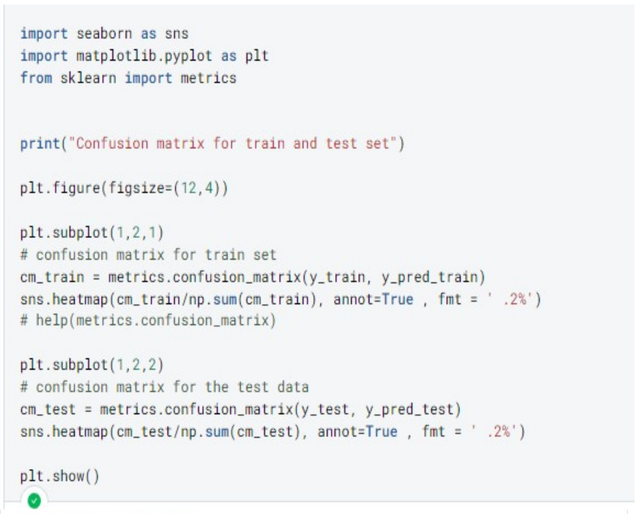
```
Time Taken: 0.02 seconds
```

```
# Prediction Train Data
y_pred_train = mnb.predict(X_train_transformed)

print("Naive Bayes accuracy", accuracy_score(y_pred_train , y_train))
print(classification_report(y_pred_train , y_train))
```

```
Naive Bayes accuracy 0.983101499673984
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     18952
           1       0.97      1.00      0.98     17856

    accuracy                           0.98     36808
   macro avg       0.98      0.98      0.98     36808
weighted avg       0.98      0.98      0.98     36808
```

Figure 23: Naive Bayes, Random Forest and XGBoost

Let us outline the actions we took in this section.
We utilised the oversampled data on Random Forest & Naive Bayes models after testing its performance on logistic regression.
We generated a vectorizer on the oversampled data using the 'word vectorizer.transform()' function, which was then put into the Naive Bayes model, much as we did with logistic regression.
To evaluate the model's performance, we examined three matrices: the confusion matrix, specificity, and sensitivity, that will be compared with some other models in the next section.
In the following part, we will apply the XGBoost algorithm on oversampled data.



```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics


print("Confusion matrix for train and test set")

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
# confusion matrix for train set
cm_train = metrics.confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm_train/np.sum(cm_train), annot=True , fmt = ' .2%')
# help(metrics.confusion_matrix)

plt.subplot(1,2,2)
# confusion matrix for the test data
cm_test = metrics.confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm_test/np.sum(cm_test), annot=True , fmt = ' .2%')

plt.show()
```

```
Confusion matrix for train and test set
```

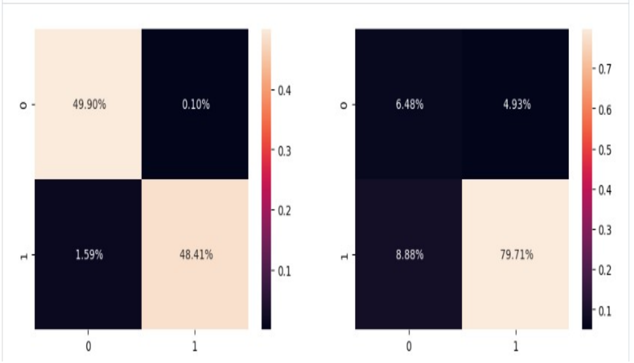
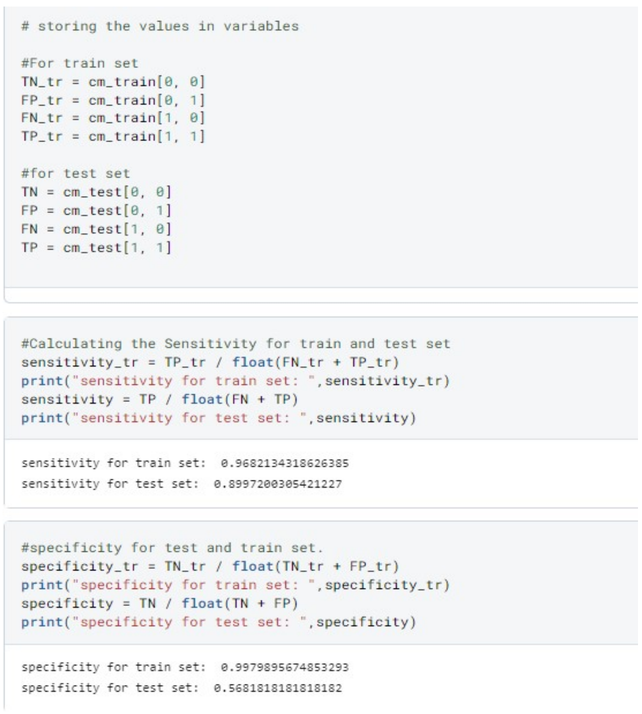


Figure 24: Conclusion metrix for train and test set

Till now, we have implemented and evaluated logistic regression, Naive Bayes, Random Forest, and XGBoost classification models. In the following section, we will utilise hyperparameter tweaking to even further increase the performance of the models. Then try comparing all of them and select the best model.



```
# storing the values in variables

#For train set
TN_tr = cm_train[0, 0]
FP_tr = cm_train[0, 1]
FN_tr = cm_train[1, 0]
TP_tr = cm_train[1, 1]

#for test set
TN = cm_test[0, 0]
FP = cm_test[0, 1]
FN = cm_test[1, 0]
TP = cm_test[1, 1]
```

```
#Calculating the Sensitivity for train and test set
sensitivity_tr = TP_tr / float(FN_tr + TP_tr)
print("sensitivity for train set: ",sensitivity_tr)
sensitivity = TP / float(FN + TP)
print("sensitivity for test set: ",sensitivity)
```

```
sensitivity for train set:  0.9682134318626385
sensitivity for test set:  0.8997200305421227
```

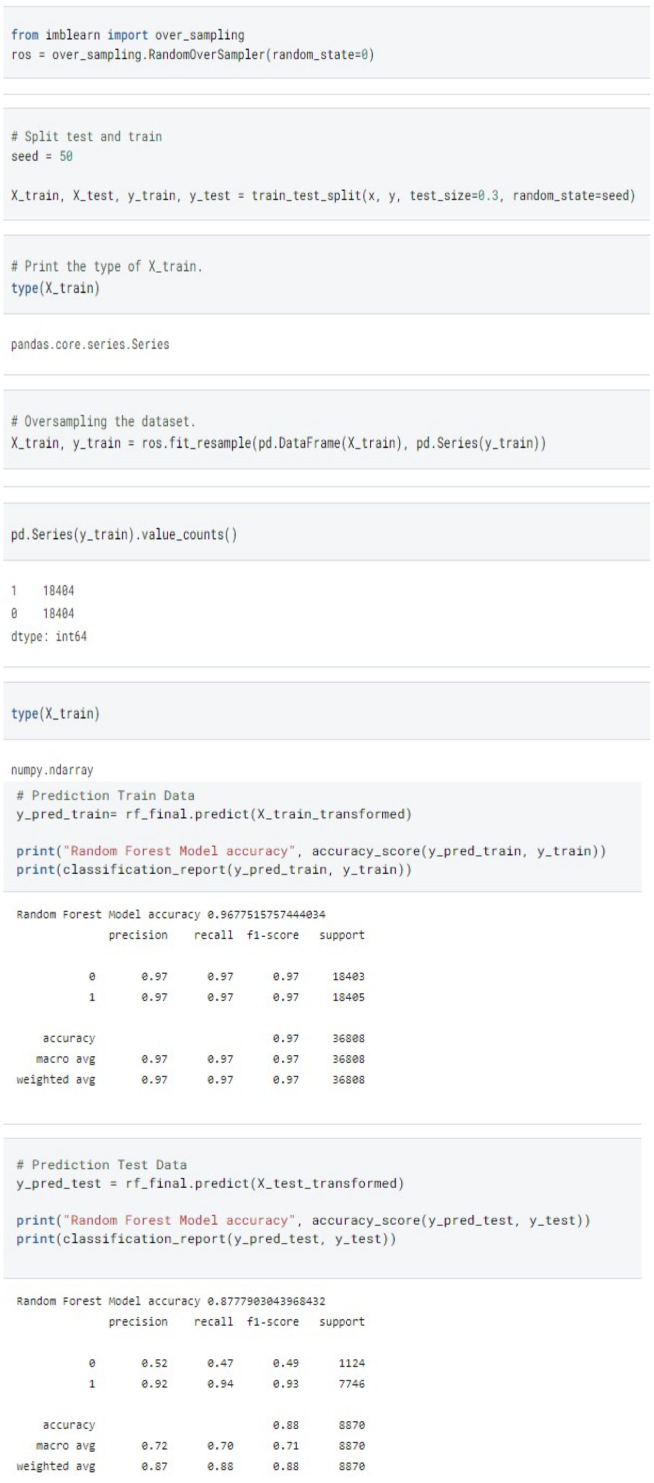
```
#specificity for test and train set.
specificity_tr = TN_tr / float(TN_tr + FP_tr)
print("specificity for train set: ",specificity_tr)
specificity = TN / float(TN + FP)
print("specificity for test set: ",specificity)
```

```
specificity for train set:  0.9979895674853293
specificity for test set:  0.5681818181818182
```

Figure 25: Specificity for Train and Test Set

XII. RANDOM FOREST AND XGBOOST_ HYPERPARAMETER TUNING

We used Naive Bayes, Random Forest, and XGBoost to oversampled data in the previous section and examined their performance matrices, specificity, and sensitivity (see figure 26, 27 and 28).



```
from imblearn import over_sampling
ros = over_sampling.RandomOverSampler(random_state=0)
```

```
# Split test and train
seed = 50

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=seed)
```

```
# Print the type of X_train.
type(X_train)
```

```
pandas.core.series.Series
```

```
# Oversampling the dataset.
X_train, y_train = ros.fit_resample(pd.DataFrame(X_train), pd.Series(y_train))
```

```
pd.Series(y_train).value_counts()
```

```
1    18404
0    18404
dtype: int64
```

```
type(X_train)
```

```
numpy.ndarray
```

```
# Prediction Train Data
y_pred_train= rf_final.predict(X_train_transformed)

print("Random Forest Model accuracy", accuracy_score(y_pred_train, y_train))
print(classification_report(y_pred_train, y_train))
```

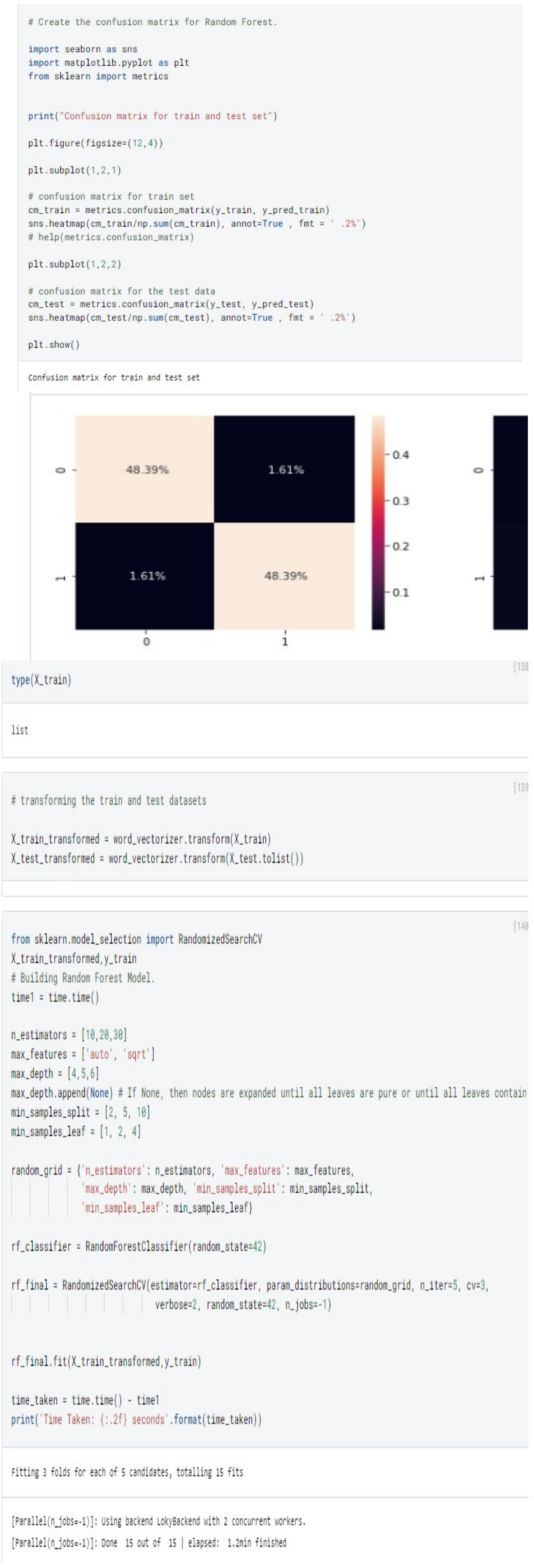
```
Random Forest Model accuracy 0.9677515757444034
              precision    recall  f1-score   support

           0       0.97      0.97      0.97     18403
           1       0.97      0.97      0.97     18405

    accuracy                           0.97     36808
   macro avg       0.97      0.97      0.97     36808
weighted avg       0.97      0.97      0.97     36808
```

```
# Prediction Test Data
y_pred_test = rf_final.predict(X_test_transformed)

print("Random Forest Model accuracy", accuracy_score(y_pred_test, y_test))
print(classification_report(y_pred_test, y_test))
```

```
Random Forest Model accuracy 0.8777903043968432
              precision    recall  f1-score   support

           0       0.52      0.47      0.49      1124
           1       0.92      0.94      0.93      7746

    accuracy                           0.88      8870
   macro avg       0.72      0.70      0.71      8870
weighted avg       0.87      0.88      0.88      8870
```

Figure 26: Random Forest and XGBoost Hyperparameter Tuning

We will discover how to create a Random Forest model using hyperparameter tweaking in the next section.



```
# Create the confusion matrix for Random Forest.

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics


print("Confusion matrix for train and test set")

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)

# confusion matrix for train set
cm_train = metrics.confusion_matrix(y_train, y_pred_train)
sns.heatmap(cm_train/np.sum(cm_train), annot=True , fmt = ' .2%')
# help(metrics.confusion_matrix)

plt.subplot(1,2,2)

# confusion matrix for the test data
cm_test = metrics.confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm_test/np.sum(cm_test), annot=True , fmt = ' .2%')

plt.show()
```

```
Confusion matrix for train and test set
```

```
[138
type(X_train)
```

```
list
```

```
[139
# transforming the train and test datasets

X_train_transformed = word_vectorizer.transform(X_train)
X_test_transformed = word_vectorizer.transform(X_test.tolist())
```

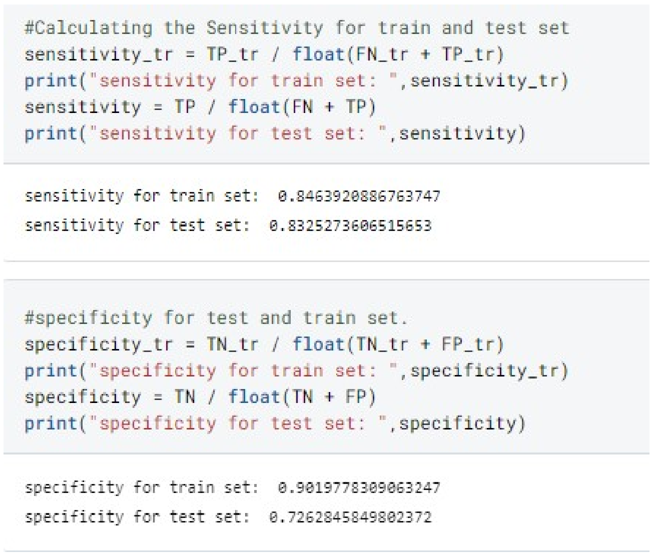
```
[140
from sklearn.model_selection import RandomizedSearchCV
X_train_transformed,y_train
# Building Random Forest Model.
time1 = time.time()

n_estimators = [10,20,30]
max_features = ['auto', 'sqrt']
max_depth = [4,5,6]
max_depth.append(None) # If None, then nodes are expanded until all leaves are pure or until all leaves contain
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]

random_grid = {'n_estimators': n_estimators, 'max_features': max_features,
               'max_depth': max_depth, 'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}

rf_classifier = RandomForestClassifier(random_state=42)

rf_final = RandomizedSearchCV(estimator=rf_classifier, param_distributions=random_grid, n_iter=5, cv=3,
                              verbose=2, random_state=42, n_jobs=-1)


rf_final.fit(X_train_transformed,y_train)

time_taken = time.time() - time1
print('Time Taken: {:.2f} seconds'.format(time_taken))
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed:  1.2min finished
```

Figure 27: ML – hyperparameter tweaking

As we now know, the optimal set of parameters for a model may be obtained by using the hyperparameter tuning approach.
Here is a brief summary of how the oversampled data was subjected to hyperparameter tuning:
In general, there are two methods for adjusting hyperparameters: "randomised search" and "grid search." To find the ideal parameters for this final project, we utilised a randomised search.
To achieve this, we first set the parameter space that the algorithm would use to find the optimal parameters within that range before importing RandomizedSearchCV from'sklearn.model selection'.
One thing to bear in mind is that finding the ideal settings will take longer the more parameter combinations are taken into account.



```
#Calculating the Sensitivity for train and test set
sensitivity_tr = TP_tr / float(FN_tr + TP_tr)
print("sensitivity for train set: ",sensitivity_tr)
sensitivity = TP / float(FN + TP)
print("sensitivity for test set: ",sensitivity)
```

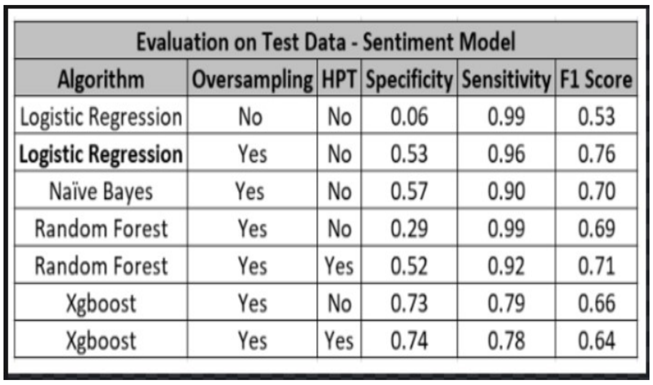
```
sensitivity for train set:  0.8463920886763747
sensitivity for test set:  0.8325273606515653
```

```
#specificity for test and train set.
specificity_tr = TN_tr / float(TN_tr + FP_tr)
print("specificity for train set: ",specificity_tr)
specificity = TN / float(TN + FP)
print("specificity for test set: ",specificity)
```

```
specificity for train set:  0.9019778309063247
specificity for test set:  0.7262845849802372
```

Figure 28: ML – Sentiment Model Summary

Table 2: General Test Scenario of ATM machine Generation

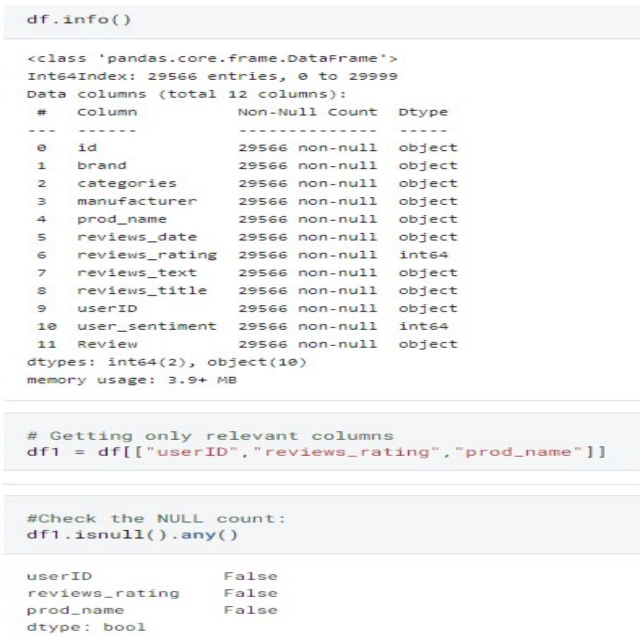


Evaluation on Test Data - Sentiment Model					
Algorithm	Oversampling	HPT	Specificity	Sensitivity	F1 Score
Logistic Regression	No	No	0.06	0.99	0.53
Logistic Regression	Yes	No	0.53	0.96	0.76
Naïve Bayes	Yes	No	0.57	0.90	0.70
Random Forest	Yes	No	0.29	0.99	0.69
Random Forest	Yes	Yes	0.52	0.92	0.71
Xgboost	Yes	No	0.73	0.79	0.66
Xgboost	Yes	Yes	0.74	0.78	0.64

We examined the table 2 that is described above in the table 1.
When the F1 score is high (tending toward 1), the model correctly predicted few false-positive and false-negative results.
Based on the table 1, it has been determined that logistic regression with oversampled data is the best model. It has an excellent F1 score, a respectable specificity, and a sensitivity of 90+. We are going to create a recommendation system in the next section.

XIII. DATA PROCESSING FOR RECOMMENDATION SYSTEM

We carried out several text processing operations and created ML models in the prior sections. Additionally, we saw that the data set we used was unbalanced(see figure 29).



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29566 entries, 0 to 29999
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              29566 non-null  object
 1   brand           29566 non-null  object
 2   categories      29566 non-null  object
 3   manufacturer    29566 non-null  object
 4   prod_name       29566 non-null  object
 5   reviews_date    29566 non-null  object
 6   reviews_rating  29566 non-null  int64
 7   reviews_text    29566 non-null  object
 8   reviews_title   29566 non-null  object
 9   userID          29566 non-null  object
 10  user_sentiment  29566 non-null  int64
 11  Review          29566 non-null  object
dtypes: int64(2), object(10)
memory usage: 3.9+ MB
```

```
# Getting only relevant columns
df1 = df[["userID","reviews_rating","prod_name"]]
```

```
#Check the NULL count:
df1.isnull().any()
```

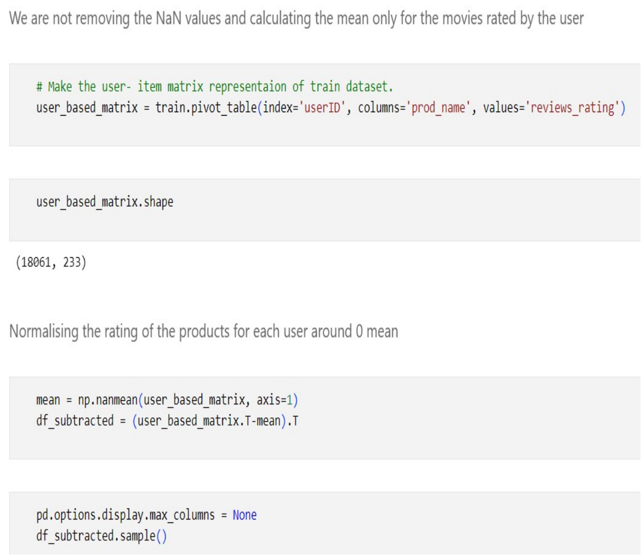
```
userID            False
reviews_rating    False
prod_name         False
dtype: bool
```

Figure 29: Data Processing

Using oversampling and hyperparameter tweaking approaches, we produced a number of ML models. We then discovered that the logistic regression model using the oversampling strategy is the best match for this specific data set among all the models.

XIV. USER-BASED RECOMMENDATION SYSTEM

We divided the data set between test and training halves in the previous phase. In this section, we will create and test a user-based recommendation system (see figure 30-37).



We are not removing the NaN values and calculating the mean only for the movies rated by the user

```
# Make the user- item matrix representaion of train dataset.
user_based_matrix = train.pivot_table(index='userID', columns='prod_name', values='reviews_rating')
```

```
user_based_matrix.shape
```

```
(18061, 233)
```

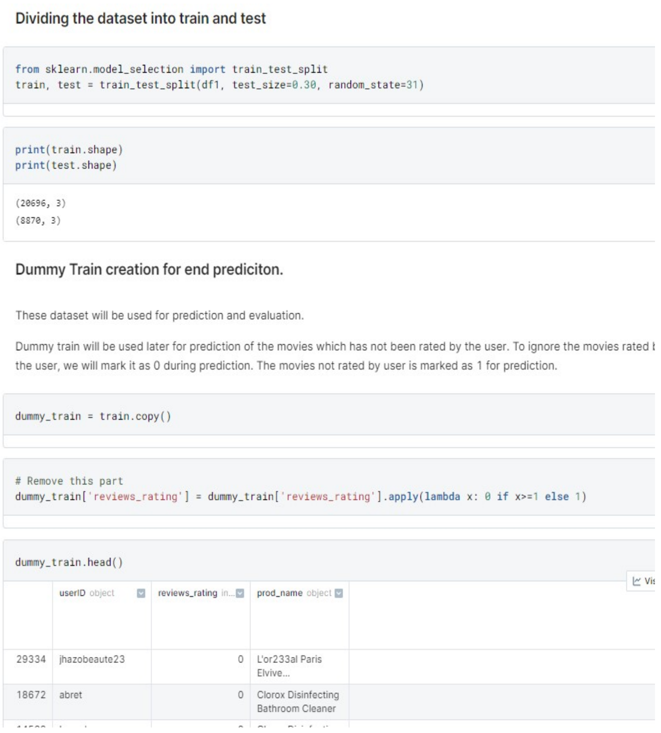
Normalising the rating of the products for each user around 0 mean

```
mean = np.nanmean(user_based_matrix, axis=1)
df_subtracted = (user_based_matrix.T-mean).T
```

```
pd.options.display.max_columns = None
df_subtracted.sample()
```

Figure 30: User-Based Recommendation System

In the following code, we will construct a user-based recommendation system and determine the user-user similarity, or correlation matrix, that will be used to propose goods to users.



Dividing the dataset into train and test

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df1, test_size=0.30, random_state=31)
```

```
print(train.shape)
print(test.shape)
```

```
(20696, 3)
(8870, 3)
```

Dummy Train creation for end prediciton.

These dataset will be used for prediction and evaluation.

Dummy train will be used later for prediction of the movies which has not been rated by the user. To ignore the movies rated b the user, we will mark it as 0 during prediction. The movies not rated by user is marked as 1 for prediction.

```
dummy_train = train.copy()
```

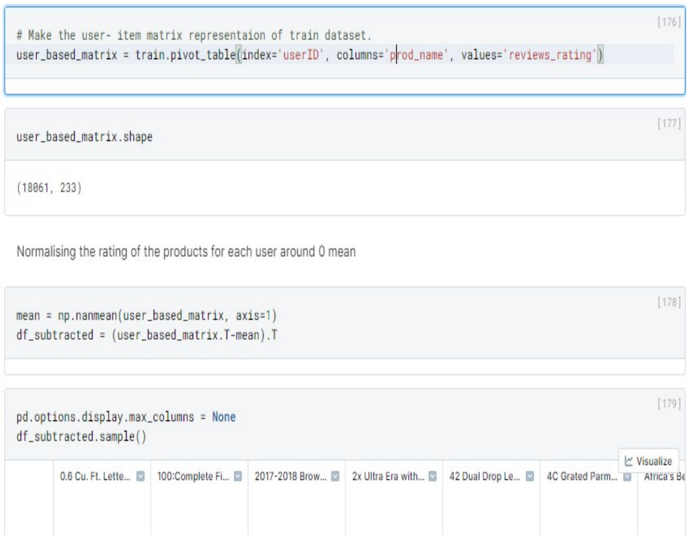
```
# Remove this part
dummy_train['reviews_rating'] = dummy_train['reviews_rating'].apply(lambda x: 0 if x>=1 else 1)
```

```
dummy_train.head()
```

	userID object	reviews_rating in...	prod_name object
29334	jhazobeaute23	0	L'or233al Paris Elvive...
18672	abret	0	Clorox Disinfecting Bathroom Cleaner

Figure 31: Dummy Train Creation End Project

The prediction and assessment processes will employ these datasets.
Later, a dummy train will be used to forecast the performance of films that the user has not yet reviewed. We shall designate the user-rated movies as 0 during prediction in order to disregard them. The user-unrated movies are indicated with a 1 for prediction



```
[176]
# Make the user- item matrix representaion of train dataset.
user_based_matrix = train.pivot_table(index='userID', columns='prod_name', values='reviews_rating')
```

```
[177]
user_based_matrix.shape
```

```
(18061, 233)
```

Normalising the rating of the products for each user around 0 mean

```
[178]
mean = np.nanmean(user_based_matrix, axis=1)
df_subtracted = (user_based_matrix.T-mean).T
```

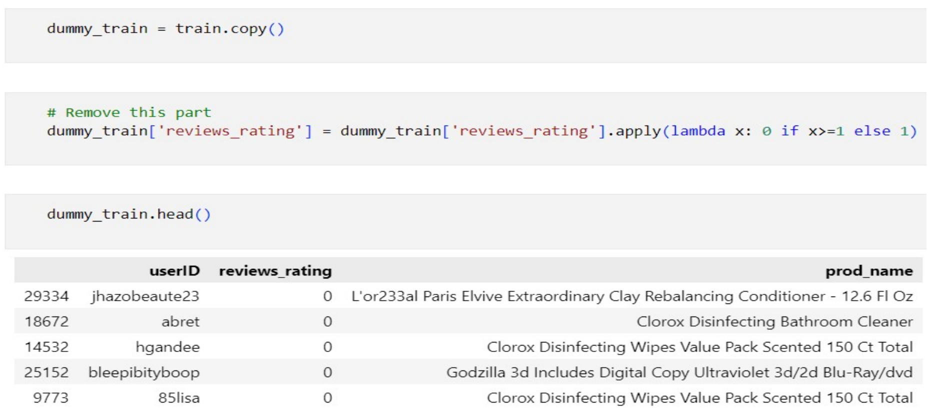
```
[179]
pd.options.display.max_columns = None
df_subtracted.sample()
```

	0.6 Cu. Ft. Lette...	100:Complete Fi...	2017-2018 Brow...	2x Ultra Era with...	42 Dual Drop Le...	4C Grated Parm...	Africa's Be

Figure 32: Normalising the Rating of Product

Using the modified Cosine
We do not eliminate the NaN values and merely compute the mean for the user-rated movies.
Now let's examine the system of product recommendations

we are going to construct. We will divide the data set into test and training data sets in the next section.



```
dummy_train = train.copy()
```

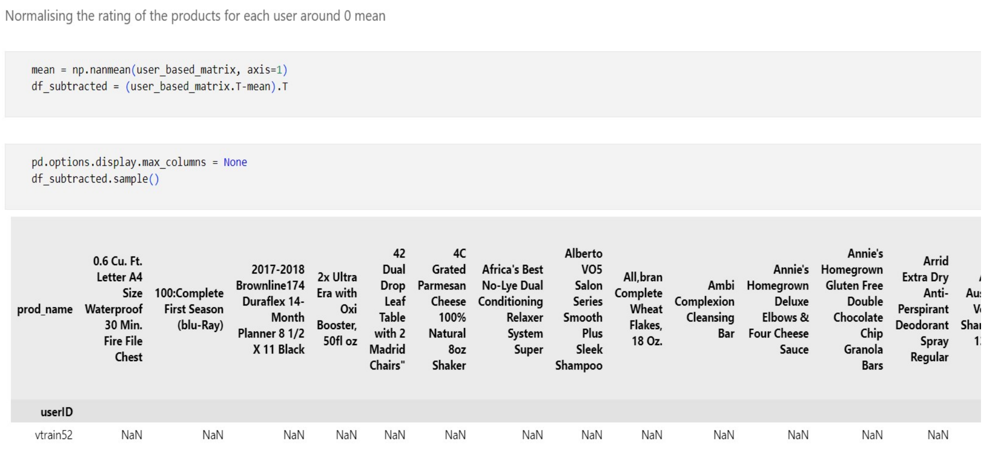
```
# Remove this part
dummy_train['reviews_rating'] = dummy_train['reviews_rating'].apply(lambda x: 0 if x>=1 else 1)
```

```
dummy_train.head()
```

	userID	reviews_rating	prod_name
29334	jhazobeaute23	0	L'or233al Paris Elvive Extraordinary Clay Rebalancing Conditioner - 12.6 Fl Oz
18672	abret	0	Clorox Disinfecting Bathroom Cleaner
14532	hgandee	0	Clorox Disinfecting Wipes Value Pack Scented 150 Ct Total
25152	bleepibityboop	0	Godzilla 3d Includes Digital Copy Ultraviolet 3d/2d Blu-Ray/dvd
9773	85lisa	0	Clorox Disinfecting Wipes Value Pack Scented 150 Ct Total

Figure 33: Separation of the entire data set into the test and train data sets

As can see in this code how we separated the entire data set into the test and train data sets.
The test data set will be used to assess the recommendation model while the train data set will be utilised to train the cosine similarity matrix between the users and items.



Normalising the rating of the products for each user around 0 mean

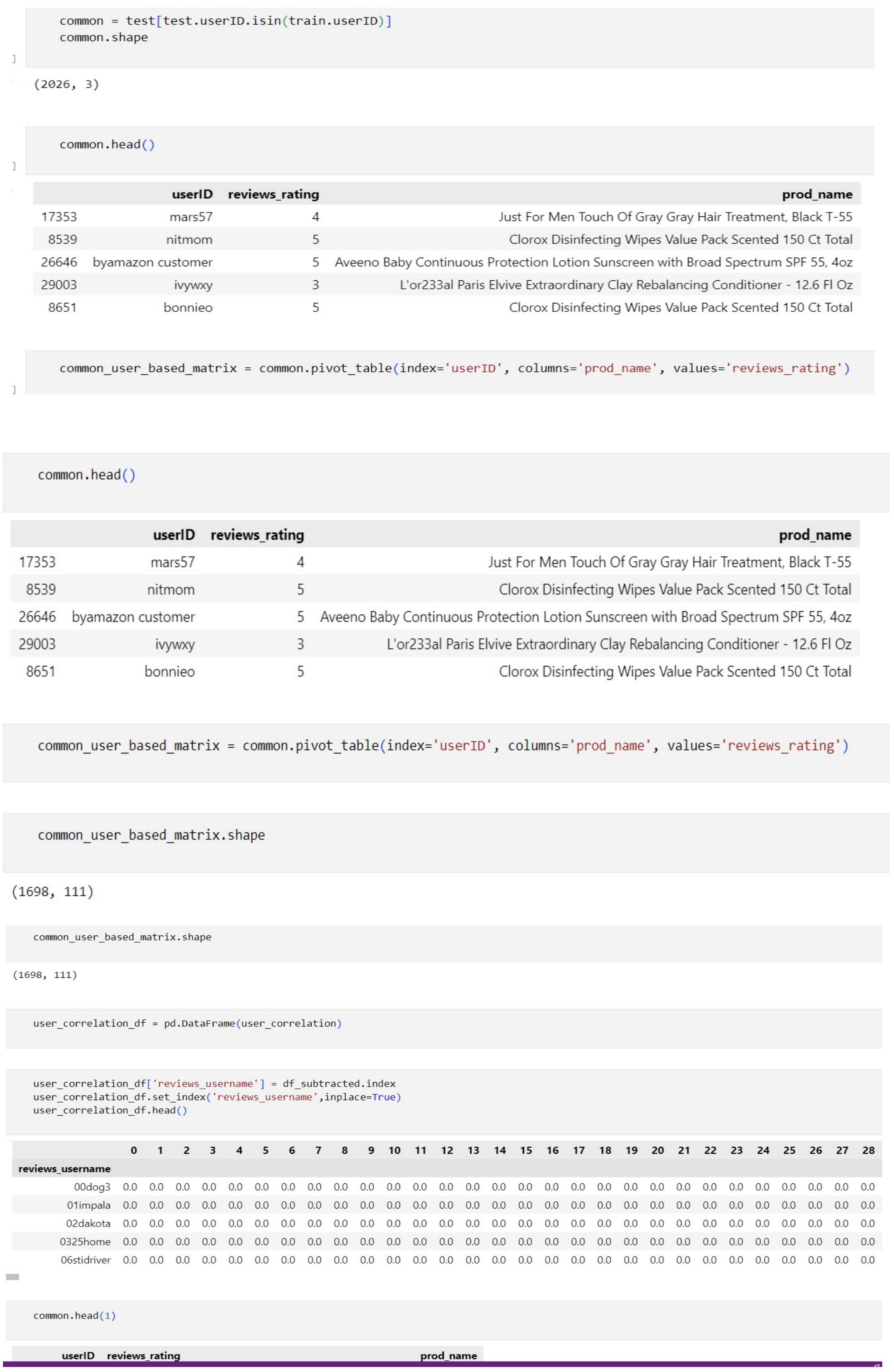
```
mean = np.nanmean(user_based_matrix, axis=1)
df_subtracted = (user_based_matrix.T-mean).T
```

```
pd.options.display.max_columns = None
df_subtracted.sample()
```

prod_name	0.6 Cu. Ft. Letter A4 Size Waterproof 30 Min. Fire File Chest	100:Complete First Season (blu-Ray)	2017-2018 Brownline174 Duraflex 14-Month Planner 8 1/2 X 11 Black	2x Ultra Era with Oxi Booster, 50fl oz	42 Dual Drop Leaf Table with 2 Madrid Chairs"	4C Grated Parmesan Cheese 100% Natural 8oz Shaker	Africa's Best No-Lye Dual Conditioning Relaxer System Super	Alberto VO5 Salon Series Smooth Plus Sleek Shampoo	All,bran Complete Wheat Flakes, 18 Oz.	Ambi Complexion Cleansing Bar	Annie's Homegrown Deluxe Elbows & Four Cheese Sauce	Annie's Homegrown Gluten Free Double Chocolate Chip Granola Bars	Arrid Extra Dry Anti-Perspirant Deodorant Spray Regular
userID													
vtrain52	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 34: Normalising the rating of the product for each user around 0 mean

Additionally, we built a data set called "dummy train" that was later transformed into a matrix format using the "pivot table" function. All the blank numbers were tagged as 0, while all the other rating values were marked as 1. To obtain the anticipated ratings only in the final predicted matrix, the "dummy train" was developed.
The following actions were taken in this code:
Using the pivot table function, we transformed the train data set to matrix format. The 'user based matrix' in this case was in the order of 18,061 X 233, where 18061 is the number of users in the train data set and 233 is the number of unique items in the train data set.
After building the 'user based matrix,' we normalised the product ratings for each user to a mean of 0.
We computed the 'user correlation' matrix using the 'pairwise distances' function to determine user correlation.
The train dataset had 18061 users, as we could see. As a result, the order of the user correlation matrix is 18,061 X 18,061.
When we identified the user correlation matrix, we put a 0 whenever there was a negative correlation between the users.
Finally, we computed a dot product of the 'user correlation' matrix and the 'user based matrix' to obtain the predicted ratings of the items for each user and recorded the results as 'user predicted ratings.'
Because we are only interested in items that have not been evaluated by users, we must disregard products that have been rated by users by setting it to zero. To do this, multiply 'user predicted ratings' by 'dummy train' to obtain the 'user final rating'. Only anticipated ratings of users relating to items not rated by the user will be included in the 'user final rating' matrix.



```
common = test[test.userID.isin(train.userID)]
common.shape
```

```
(2026, 3)
```

```
common.head()
```

	userID	reviews_rating	prod_name
17353	mars57	4	Just For Men Touch Of Gray Gray Hair Treatment, Black T-55
8539	nitmom	5	Clorox Disinfecting Wipes Value Pack Scented 150 Ct Total
26646	byamazon customer	5	Aveeno Baby Continuous Protection Lotion Sunscreen with Broad Spectrum SPF 55, 4oz
29003	ivywxy	3	L'or233al Paris Elvive Extraordinary Clay Rebalancing Conditioner - 12.6 Fl Oz
8651	bonnieo	5	Clorox Disinfecting Wipes Value Pack Scented 150 Ct Total

```
common_user_based_matrix = common.pivot_table(index='userID', columns='prod_name', values='reviews_rating')
```

```
common.head()
```

	userID	reviews_rating	prod_name
17353	mars57	4	Just For Men Touch Of Gray Gray Hair Treatment, Black T-55
8539	nitmom	5	Clorox Disinfecting Wipes Value Pack Scented 150 Ct Total
26646	byamazon customer	5	Aveeno Baby Continuous Protection Lotion Sunscreen with Broad Spectrum SPF 55, 4oz
29003	ivywxy	3	L'or233al Paris Elvive Extraordinary Clay Rebalancing Conditioner - 12.6 Fl Oz
8651	bonnieo	5	Clorox Disinfecting Wipes Value Pack Scented 150 Ct Total

```
common_user_based_matrix = common.pivot_table(index='userID', columns='prod_name', values='reviews_rating')
```

```
common_user_based_matrix.shape
```

```
(1698, 111)
```

```
common_user_based_matrix.shape
```

```
(1698, 111)
```

```
user_correlation_df = pd.DataFrame(user_correlation)
```

```
user_correlation_df['reviews_username'] = df_subtracted.index
user_correlation_df.set_index('reviews_username',inplace=True)
user_correlation_df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
reviews_username																													
00dog3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
01impala	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
02dakota	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0325home	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
06stidriver	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

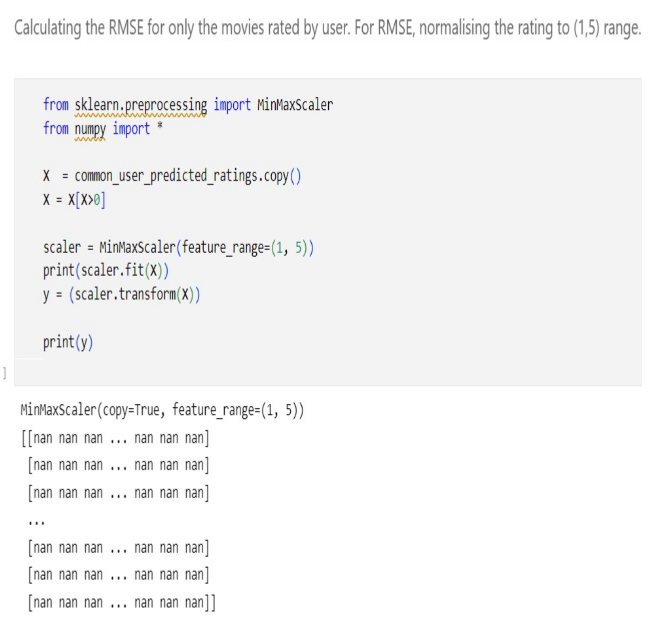
```
common.head(1)
```

	userID	reviews_rating	prod_name

Figure 35: Dot product of the 'user correlation' matrix and the 'user-based matrix'

We will evaluate the recommendation system using the test data set in the next part.
Let's condense the lessons we've learned from this code into the following points:

It's crucial to bear in mind that we must exclude individuals who are exclusive to the test data set and retain only those users who are present in both the test and train data sets.



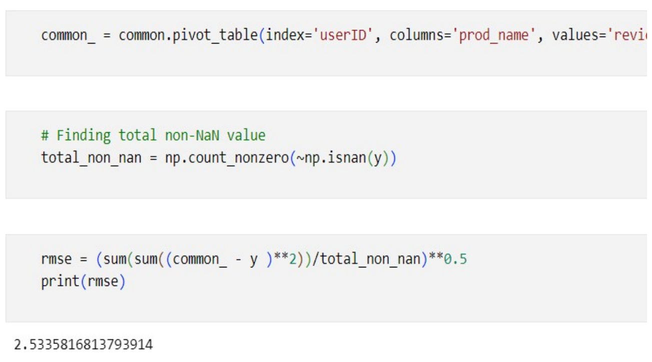
Calculating the RMSE for only the movies rated by user. For RMSE, normalising the rating to (1,5) range.

```
from sklearn.preprocessing import MinMaxScaler
from numpy import *

X  = common_user_predicted_ratings.copy()
X = X[X>0]

scaler = MinMaxScaler(feature_range=(1, 5))
print(scaler.fit(X))
y = (scaler.transform(X))

print(y)
```

```
MinMaxScaler(copy=True, feature_range=(1, 5))
[[nan nan nan ... nan nan nan]
 [nan nan nan ... nan nan nan]
 [nan nan nan ... nan nan nan]
 ...
 [nan nan nan ... nan nan nan]
 [nan nan nan ... nan nan nan]
 [nan nan nan ... nan nan nan]]
```

Figure 36: Calculating the RMSE

We have now reduced the number of users to those who appear in both the test and train data sets, and we have developed a product called "common user based matrix" that contains a user matrix. There are 1,698 users and 111 items, according to the common user based matrix matrix, which has an order of 1,698 X 111.



```
common_ = common.pivot_table(index='userID', columns='prod_name', values='revi
```

```
# Finding total non-NaN value
total_non_nan = np.count_nonzero(~np.isnan(y))
```

```
rmse = (sum(sum((common_ - y )**2))/total_non_nan)**0.5
print(rmse)
```

```
2.5335816813793914
```

Figure 37: Finding total non-NaN value

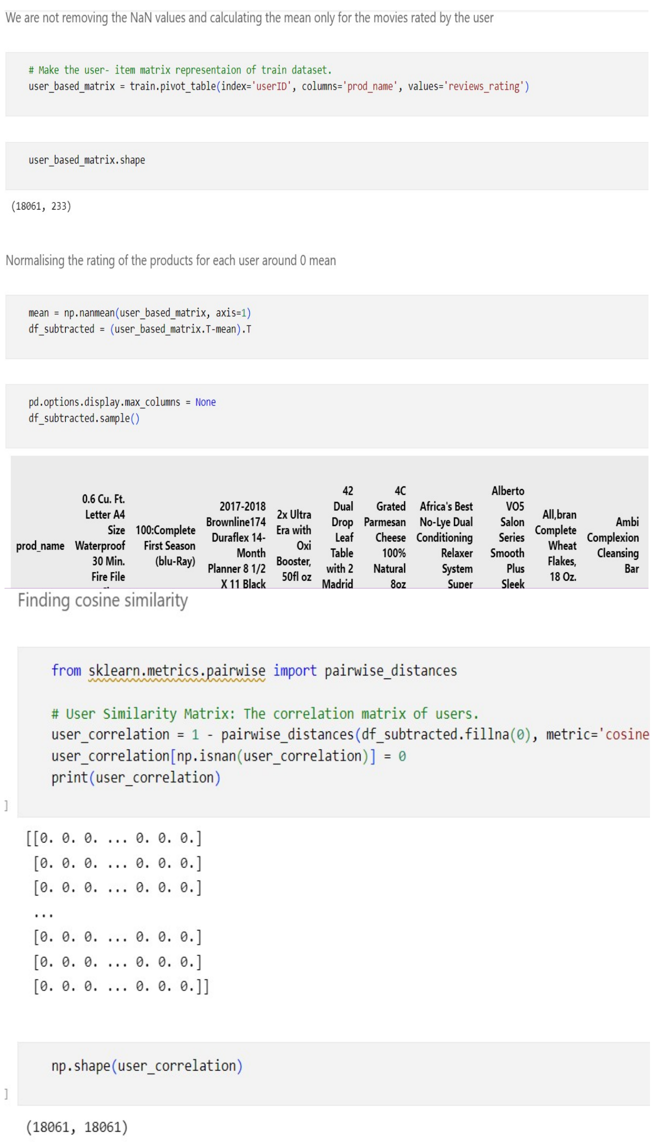
In the subsequent code, we will run the trained model's assessment and extract the correlation coefficient for typical users.

Let's review what we discovered in the code:

- The user correlation matrix that we acquired from the train data set must first have the correlation coefficient of the common users extracted from it. As a result, we receive the 1698 x 1698 user correlation matrix known as "user correlation df 3" for the common users.
- We must now take the dot product of the "user correlation df 3" matrix and the "common user based matrix" in order to forecast the test users' ratings.
- We may only complete the evaluation on the goods that have received user ratings. This is thus rated as 1. In contrast to "dummy train," this.
- A matrix of the kind 1,698 X 111 will result from entering the values 1 and 0 in the "dummy test" matrix.
- For a prognosis on the products that users have rated, multiply the "common user expected ratings" matrix by the "dummy test" matrix.
- The RMSE (Root Mean Square Error), which in the user-based recommendation system is 2.533, may be calculated after we know both the anticipated and actual ratings.
- We'll design and evaluate an item-based recommendation system in the part after that.

XV. ITEM-BASED RECOMMENDATION SYSTEM

We will now create an item-based recommendation system after creating a user-based recommendation system in the previous part (see figure 38-39).



We are not removing the NaN values and calculating the mean only for the movies rated by the user

```
# Make the user- item matrix representaion of train dataset.
user_based_matrix = train.pivot_table(index='userID', columns='prod_name', values='reviews_rating')
```

```
user_based_matrix.shape
```

```
(18061, 233)
```

Normalising the rating of the products for each user around 0 mean

```
mean = np.nanmean(user_based_matrix, axis=1)
df_subtracted = (user_based_matrix.T-mean).T
```

```
pd.options.display.max_columns = None
df_subtracted.sample()
```

prod_name	0.6 Cu. Ft. Letter A4 Size Waterproof 30 Min. Fire File	100:Complete First Season (blu-Ray)	2017-2018 Brownline174 Duraflex 14-Month Planner 8 1/2 X 11 Black	2x Ultra Era with Oxi Booster, 50fl oz	42 Dual Drop Leaf Table with 2 Madrid	4C Grated Parmesan Cheese 100% Natural 8oz	Africa's Best No-Lye Dual Conditioning Relaxer System Super	Alberto VO5 Salon Series Smooth Plus Sleek	All,bran Complete Wheat Flakes, 18 Oz.	Ambi Complexion Cleansing Bar

Finding cosine similarity

```
from sklearn.metrics.pairwise import pairwise_distances

# User Similarity Matrix: The correlation matrix of users.
user_correlation = 1 - pairwise_distances(df_subtracted.fillna(0), metric='cosine
user_correlation[np.isnan(user_correlation)] = 0
print(user_correlation)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

```
np.shape(user_correlation)
```

```
(18061, 18061)
```

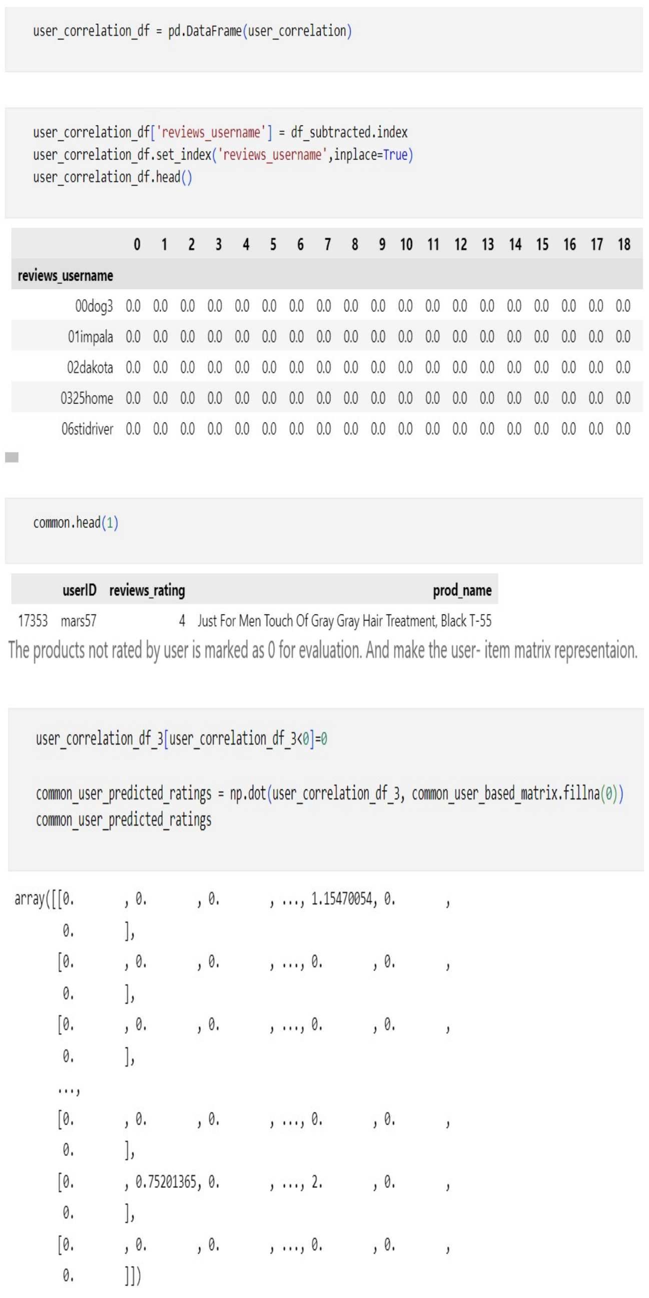
Figure 38: Item-based recommendation system

The procedures are identical to what we already know from the prior lesson. So, in the next section of code, let's begin creating the item-based recommendation system.

We took the actions shown below in the code previously mentioned:

We started by making a "item based matrix" containing rows of items and columns of people. 233 X 18061 made up the "item based matrix," where 233 represented the number of items and 18,061 represented the number of

users.
We then computed the cosine similarity between the products using the "pairwise distances" function, arriving with a "item correlation" matrix with dimensions of 233 X 233.
We computed the dot product of the "item based matrix" and the "item correlation" matrix to determine the projected ratings.
After obtaining the dot product, we limited our filtering of the reviews to those items that the customer had not rated in order to support the product's suggestion.



```
user_correlation_df = pd.DataFrame(user_correlation)
```

```
user_correlation_df['reviews_username'] = df_subtracted.index
user_correlation_df.set_index('reviews_username',inplace=True)
user_correlation_df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
reviews_username																			
00dog3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
01impala	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
02dakota	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0325home	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
06stidriver	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

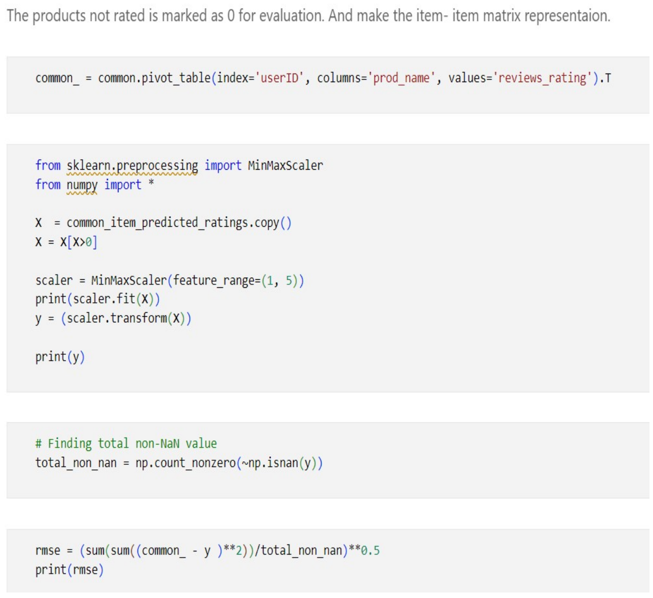
```
common.head(1)
```

	userID	reviews_rating	prod_name
17353	mars57	4	Just For Men Touch Of Gray Gray Hair Treatment, Black T-55

The products not rated by user is marked as 0 for evaluation. And make the user- item matrix representaion.

```
user_correlation_df_3[user_correlation_df_3<0]=0

common_user_predicted_ratings = np.dot(user_correlation_df_3, common_user_based_matrix.fillna(0))
common_user_predicted_ratings
```

```
array([[0.        , 0.        , 0.        , ..., 1.15470054, 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.75201365, 0.        , ..., 2.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ]])
```



The products not rated is marked as 0 for evaluation. And make the item- item matrix representaion.

```
common_ = common.pivot_table(index='userID', columns='prod_name', values='reviews_rating').T
```

```
from sklearn.preprocessing import MinMaxScaler
from numpy import *

X  = common_item_predicted_ratings.copy()
X = X[X>0]

scaler = MinMaxScaler(feature_range=(1, 5))
print(scaler.fit(X))
y = (scaler.transform(X))

print(y)
```

```
# Finding total non-NaN value
total_non_nan = np.count_nonzero(~np.isnan(y))
```

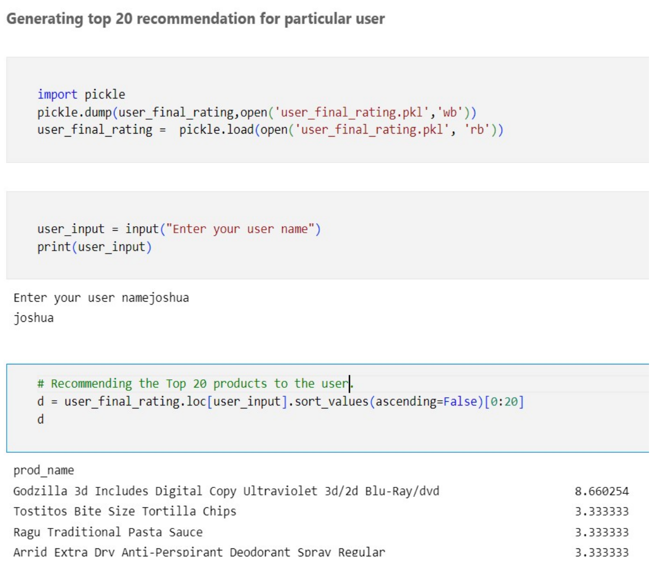
```
rmse = (sum(sum((common_ - y )**2))/total_non_nan)**0.5
print(rmse)
```

Figure 39: Item-item matrix representation

After identifying the items shared by the test and train data sets, we assessed the item-based recommendation system, just as we did with the user-based recommendation system, and obtained an RMSE value of 3.55—higher than the RMSE value we obtained for the user-based recommendation system.
We will discover how to make product recommendations to a user in the next section.

XVI. RECOMMENDING PRODUCTS TO USERS

We created both user-based and item-based recommendation systems in the past two stages.
We will discover how to give a consumer product recommendation in this section. We will employ the user-based recommendation system for this as it had a lower RMSE value than the model when we evaluated it. On the basis of the projected ratings, we gleaned from the user-based recommendation system, we will suggest the top 20 goods to a user in the following code (see figure 40-42).



Generating top 20 recommendation for particular user

```
import pickle
pickle.dump(user_final_rating,open('user_final_rating.pkl','wb'))
user_final_rating =  pickle.load(open('user_final_rating.pkl', 'rb'))
```

```
user_input = input("Enter your user name")
print(user_input)
```

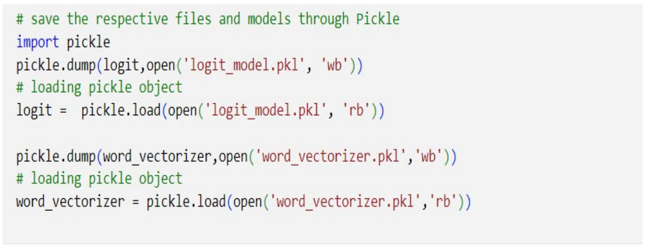
```
Enter your user namejoshua
joshua
```

```
# Recommending the Top 20 products to the user.
d = user_final_rating.loc[user_input].sort_values(ascending=False)[0:20]
d
```

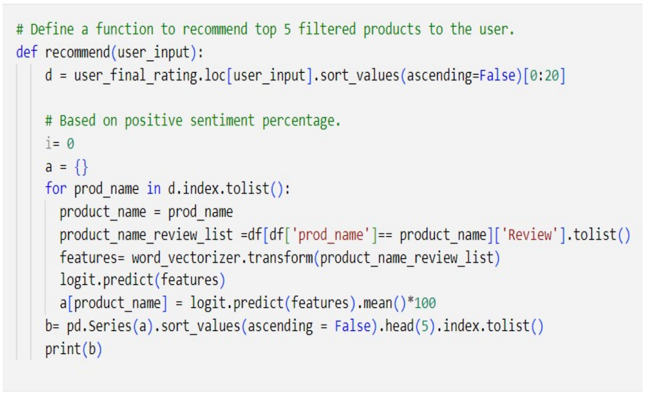
```
prod_name
Godzilla 3d Includes Digital Copy Ultraviolet 3d/2d Blu-Ray/dvd           8.660254
Tostitos Bite Size Tortilla Chips                                         3.333333
Ragu Traditional Pasta Sauce                                              3.333333
Arrid Extra Drv Anti-Perspirant Deodorant Sprav Regular                   3.333333
```

Figure 40: Recommending Products to Users

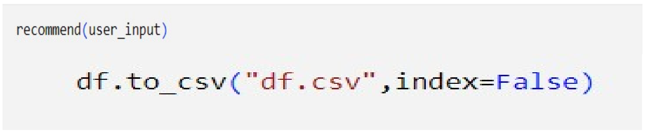
In order to anticipate the top 20 goods from the user-based recommendation system, let's attempt to grasp exactly what we need to accomplish. To obtain the anticipated ratings of the goods belonging to a user, all we need is the "user final rating" matrix.



```
# save the respective files and models through Pickle
import pickle
pickle.dump(logit,open('logit_model.pkl', 'wb'))
# loading pickle object
logit =  pickle.load(open('logit_model.pkl', 'rb'))

pickle.dump(word_vectorizer,open('word_vectorizer.pkl','wb'))
# loading pickle object
word_vectorizer = pickle.load(open('word_vectorizer.pkl','rb'))
```



```
# Define a function to recommend top 5 filtered products to the user.
def recommend(user_input):
    d = user_final_rating.loc[user_input].sort_values(ascending=False)[0:20]

    # Based on positive sentiment percentage.
    i= 0
    a = {}
    for prod_name in d.index.tolist():
      product_name = prod_name
      product_name_review_list =df[df['prod_name']== product_name]['Review'].tolist()
      features= word_vectorizer.transform(product_name_review_list)
      logit.predict(features)
      a[product_name] = logit.predict(features).mean()*100
    b= pd.Series(a).sort_values(ascending = False).head(5).index.tolist()
    print(b)
```



```
recommend(user_input)

    df.to_csv("df.csv",index=False)
```

Figure 41: "user final rating" matrix

In order to make a forecast for the deployment of the recommendation system, we must pickle the "user final rating" matrix. Once we have the user, all that is left to do is publish the top 20 goods for that user after sorting the anticipated ratings in descending order. We will discover how to improve the top 20 suggested goods using sentiment analysis and the machine learning models we created previously in the next session of code.
Let's first attempt to comprehend the principle guiding the fine-tuning:
Using the "user final rating" matrix, we have already given a user a recommendation for the top 20 goods.
Here, we are attempting to use sentiment analysis to select the best five items out of a total of twenty. We now have a list of the top 20 items from the train data set as well as reviews for each of the 20 goods. We run each product's review through the chosen logistic regression model to determine the reviewer's emotion.
Once we get the sentiment, we may determine the percentage of favorable sentiments for each product and organize them according to decreasing percentages once we have the emotion. The user's filtered suggestions would now be the top five products on this list. Let's now attempt to learn what we require in order to optimize the recommendation system. A logistic regression model to forecast the emotion associated with each product review
The review is transformed by the "word vectorizer" into the TF-IDF vectorizer before being processed by the ML model. Therefore, we must pickle the user final rating object separately from the logistic regression model and the "word vectorizer" object. We will discover how to deploy the full project using Flask and Heroku in the next section.
Deployment
To do sentiment analysis up until now, we constructed a model. Then, after fine-tuning the suggestions using the sentiment analysis model, we constructed a recommendation system and forecasted the top five goods to be suggested to a user. We will deploy the full project using Flask and Heroku in this section. Implementing the code in the Visual Studio Code environment will serve as the segment's introduction. We made an app.py file, a Flask file designed to act as a link between the HTML page and the machine learning and recommendation system models, as we saw in the code. We will use our local machine to execute the web application in the end. So we created a whole web application and installed it on a local machine. We have seen usage of Heroku to publicly launch this web application. By doing so, we developed a web application and used Flask and Heroku to deploy the complete project.

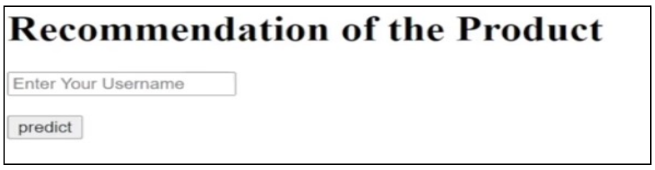


Figure 42: Deployment

XVII. CONCLUSION

We developed a recommendation engine in this research using a sentiment analysis model. The following is a summary of the procedures we followed:
One of the first and most crucial procedures in developing any ML model on textual data is text processing. Although there are a lot of stages we may add to the lexical processing of text data, we cleaned the text data by doing the following:

- Lowercasing: Lowercasing is carried out as good practise and to prevent any case sensitivity that can arise while creating a function in Python.
- Eliminating punctuation: It is a good practise to delete punctuation because it adds nothing to the process of deriving meaning from text data.
- Removing stop words: When attempting to glean meaning from text, stopwords like "the," "in," "on," and "is" are of little use. So, getting rid of them is a smart idea.
- Lemmatization: One of the fundamental phases in reducing each word to its lemma base form is lemmatization.

Following the completion of the fundamental text processing processes, we must specify the target variable and the characteristics. The user's emotion, which may be either good or negative, is unquestionably the goal variable. The word vectorizer, which we may create using a bag-of-words model, TF-IDF, or Count Vectorizer, will be one of the features.
In this project, we built the TF-IDF vectorizer and utilized it to generate the features for the classification model.
For sentiment analysis, creating classification models: We

created the logistic regression, Naive Bayes, Random Forest Classifier, and XGBoost machine learning models for classification. We could see that the data was unbalanced, thus it was necessary to do so before continuing. Out of the several combinations we tried, we decided on logistic regression with oversampling. Building the user- and item-based recommendation systems, as well as making predictions using them, followed the construction of the sentiment analysis model. These systems were then tested using the test data set. Additionally, we saw that the user-based recommendation system had a lower RMSE; as a result, we decided to utilise it to suggest goods to consumers.

The user-based recommendation system was the one we selected for fine-tuning. We then refined the suggestions based on the product reviews and its expected feelings using the sentiment analysis model.

Final step: Using Flask and Heroku, we deployed the entire project.

REFERENCES

[1] E. Turban, D. King, J. Lee and D. Viehland, Electronic Commerce: A Managerial Perspective, Upper Saddle River, NJ, USA: Prentice-Hall, 2002.

[2] Liu, J.G., Zhou, T., Wang, B.H.: Research Progress of Personalized Recommendation System. Progress in Natural Science 19(1), 1–15 (2009)

[3] J.B. Schafer, J. Konstan and J. Riedl, "Recommender systems in e-commerce", Proceedings of the 1st ACM conference on Electronic commerce, pp. 158-166, 1999, November.

[4] M.B. Dias, D. Locher, M. Li, W. El-Deredy and P.J. Lisboa, "The value of personalised recommender systems to e-business: a case study", Proceedings of the 2008 ACM conference on Recommender systems, pp. 291-294, 2008, October.

[5] Recommendation Systems: Applications Examples & Benefits, 2020, [online] Available: https://research.aimultiple.com/recommendation-system/#media.

[6] M. Sree Vani, "IJARCCE A Recommender System for Online Advertising", International Journal of Advanced Research in Computer and Communication Engineering, vol. 5, no. 2, 2016.

[7] Z. Fayyaz, M. Ebrahimian, D. Nawara, A. Ibrahim and R. Kashef, "Recommendation Systems: Algorithms Challenges Metrics and Business Opportunities", Applied Sciences, vol. 10, no. 21, pp. 7748, 2020.

[8] Linden, Greg, Brent Smith, and Jeremy York. "Amazon.com recommendations: Item-to-item collaborative filtering." IEEE Internet computing 7.1 (2003): 76-80

[9] Park, Sung Eun, Sangkeun Lee, and Sang-goo Lee. "Session-based collaborative filtering for predicting the next song." Computers, Networks, Systems and Industrial Engineering (CNSI), 2011 First ACIS/JNU International Conference on. IEEE, 2011.

[10] Suksawatchon Ureerat, Sumet Darapisut, and Jakkarin Suksawatchon. "Incremental session based collaborative filtering with forgetting mechanisms." 2015 International Computer Science and Engineering Conference (ICSEC). IEEE, 2015.

[11] Mustafa, Ghulam, and Ingo Frommholz. "Performance comparison of top N recommendation algorithms." 2015 Fourth International Conference on Future Generation Communication Technology (FGCT). IEEE, 2015.

[12] Oard, Douglas W., and Jinmook Kim. "Implicit feedback for recommender systems." Proceedings of the AAAI workshop on recommender systems. 1998.

[13] Romadhony, Ade, Said Al Faraby, and Bambang Pudjoatmodjo. "Online shopping recommender system using hybrid method." Information and Communication Technology (ICoICT), 2013 International Conference of. IEEE, 2013.

ABOUT THE AUTHORS

Muzakkiruddin Ahmed Mohammed is a B.Tech student in the Department of Electrical and Electronics Engineering, Lords Institute of Engineering and Technology, Hyderabad, India