SENTIMENTAL ANALYIS USING AMAZON REVIEWS

SUBJECT: BIG DATA TOOLS PROFESSOR: ZAFIR SYED

TEAM:
BABITA MALHOTRA
ARAVIND YELLANKI
DEVI SAMYUKTHA CHITTURI

PROJECT OVERVIEW

SENTIMENTAL ANALYSIS?

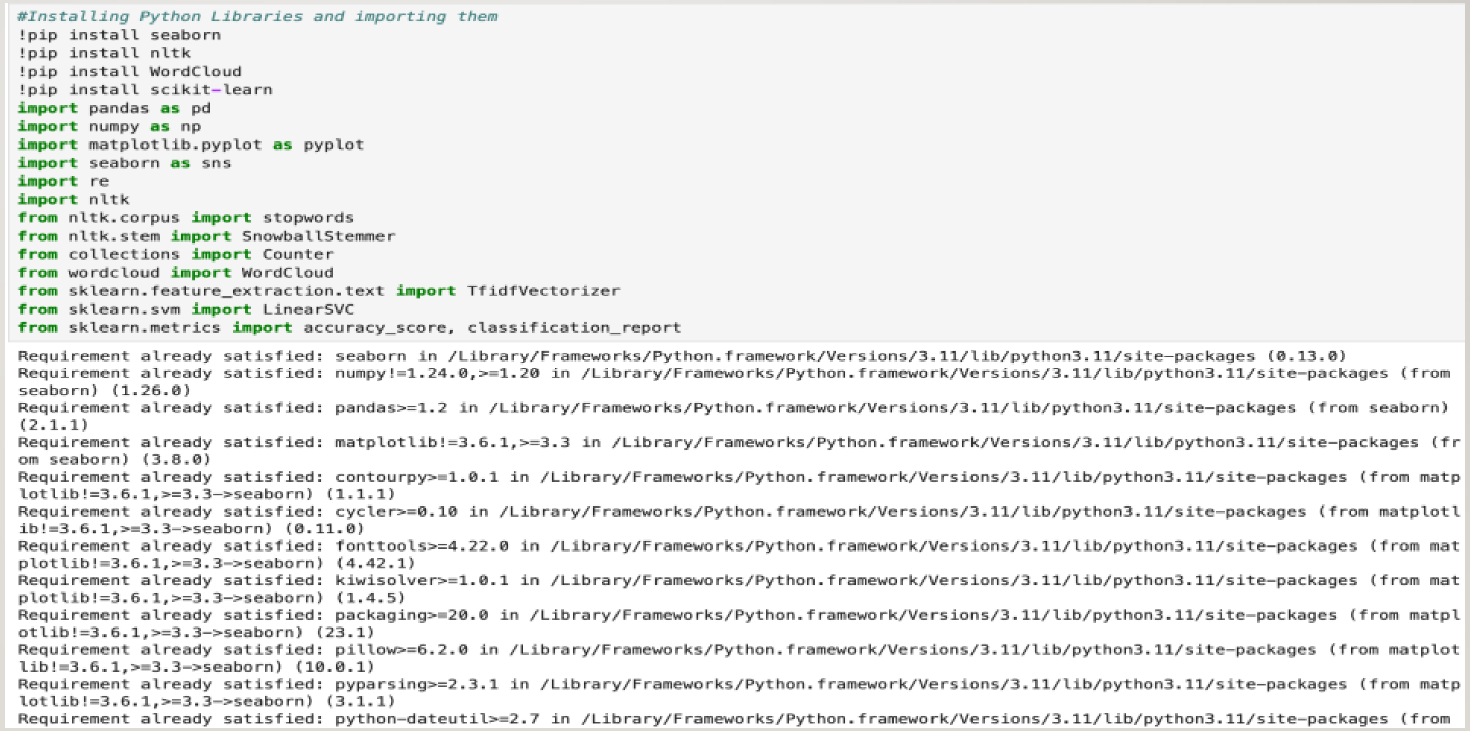
Sentimental Analysis uses Natural Language processing to identify the emotions behind the text. Organizations can utilize this to make data driven decisions based on user input and public opinion by using it to extract actionable insights from massive amounts of textual data. The main objective is to examine and comprehend the opinions stated in Amazon product reviews. The goal of this analysis could be to glean information on client sentiment trends, satisfaction levels, and opinions.

DATA ACQUISITION

We extracted the dataset from the KAGGLE, the dataset consists of reviews from Amazon. The dataset comprises 35 million reviews from above 6 million users on slightly above 2 million products over the time span of 18 years from Stanford Network Analysis Project(SNAP). The dataset has a size of 1.5 gigabytes. The dataset consists of 1,800,000 training samples. WHY ONLY THIS? As the humans are revolving around the easy access of online services such as Amazon, Uber, Uber eats, Lyft etc. The priority of opting a new service will be based on the reviews. This drives us to work on Amazon Reviews. The numerous products among the various categories makes it possible to conduct through analysis across the different industries.

ENVIRONMENT SETUP

First, set up the Python Environment that includes packages for Data Analysis, including PySpark, WordCloud, NLTK and then importing them into the script.

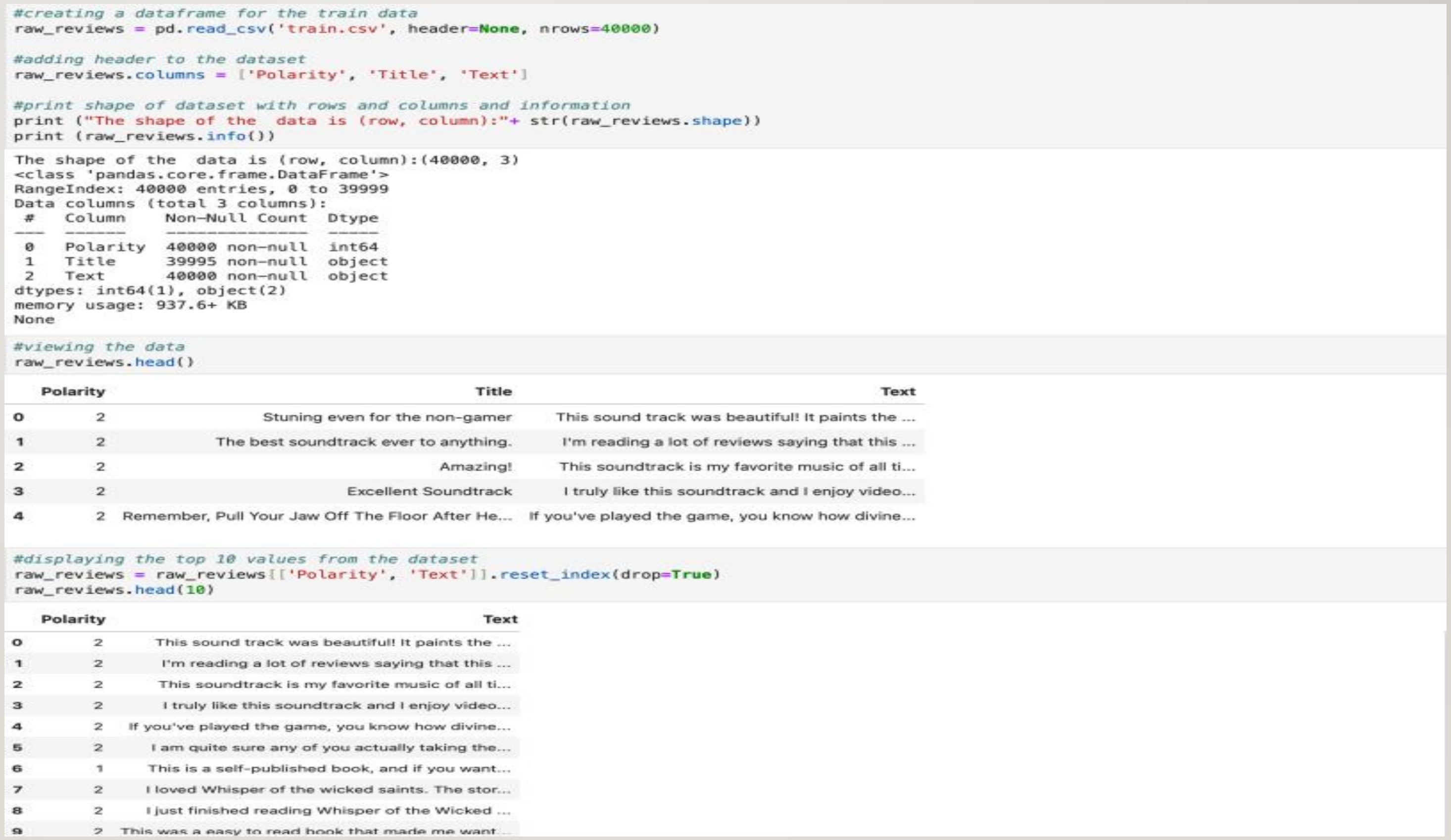


```
#Installing Python Libraries and importing them
!pip install seaborn
!pip install nltk
!pip install WordCloud
!pip install scikit-learn
import pandas as pd
import numpy as np
import matplotlib.pyplot as pyplot
import seaborn as sns
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from collections import Counter
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report
```

```
Requirement already satisfied: seaborn in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (0.13.0)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from
seaborn) (1.26.0)
Requirement already satisfied: pandas>=1.2 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from seaborn)
(2.1.1)
Requirement already satisfied: matplotlib!=3.6.1,>=3.3 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (fr
om seaborn) (3.8.0)
Requirement already satisfied: contourpy>=1.0.1 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from matp
lotlib!=3.6.1,>=3.3->seaborn) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from matplotl
ib!=3.6.1,>=3.3->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from mat
plotlib!=3.6.1,>=3.3->seaborn) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from mat
plotlib!=3.6.1,>=3.3->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from matpl
otlib!=3.6.1,>=3.3->seaborn) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from matplot
lib!=3.6.1,>=3.3->seaborn) (10.0.1)
Requirement already satisfied: pyparsing>=2.3.1 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from matp
lotlib!=3.6.1,>=3.3->seaborn) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from
```

LOADING THE DATASET

Loaded the train dataset into the created data frame, added headers to the dataset for data readable. Here, we mostly used the 'Polarity' and 'Text' columns for the data analysis. Read the top 10 columns of the data set.



```
#creating a dataframe for the train data
raw_reviews = pd.read_csv('train.csv', header=None, nrows=40000)

#adding header to the dataset
raw_reviews.columns = ['Polarity', 'Title', 'Text']

#print shape of dataset with rows and columns and information
print ("The shape of the  data is (row, column):"+ str(raw_reviews.shape))
print (raw_reviews.info())
```

```
The shape of the  data is (row, column):(40000, 3)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40000 entries, 0 to 39999
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Polarity  40000 non-null  int64 
 1   Title     39995 non-null  object
 2   Text      40000 non-null  object
dtypes: int64(1), object(2)
memory usage: 937.6+ KB
None
```

```
#viewing the data
raw_reviews.head()
```

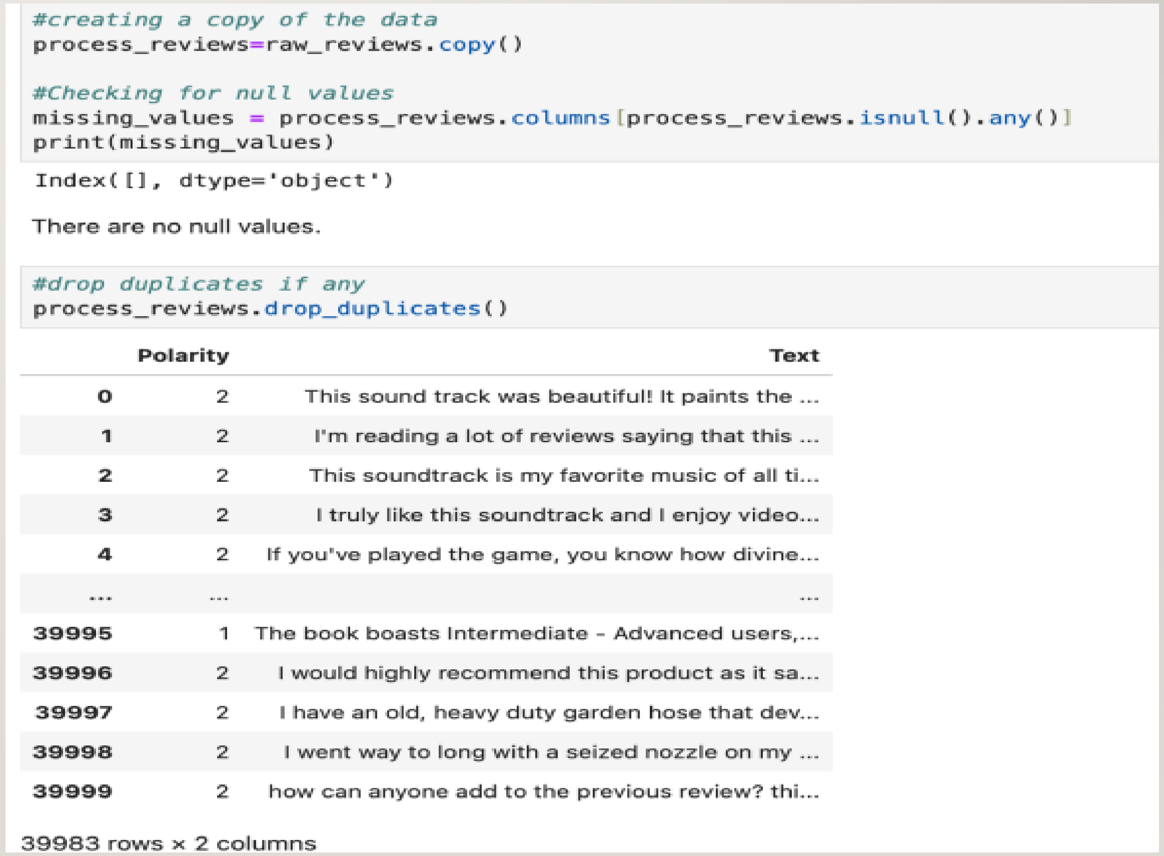
	Polarity	Title	Text
0	2	Stuning even for the non-gamer	This sound track was beautiful! It paints the ...
1	2	The best soundtrack ever to anything.	I'm reading a lot of reviews saying that this ...
2	2	Amazing!	This soundtrack is my favorite music of all ti...
3	2	Excellent Soundtrack	I truly like this soundtrack and I enjoy video...
4	2	Remember, Pull Your Jaw Off The Floor After He...	If you've played the game, you know how divine...

```
#displaying the top 10 values from the dataset
raw_reviews = raw_reviews[['Polarity', 'Text']].reset_index(drop=True)
raw_reviews.head(10)
```

	Polarity	Text
0	2	This sound track was beautiful! It paints the ...
1	2	I'm reading a lot of reviews saying that this ...
2	2	This soundtrack is my favorite music of all ti...
3	2	I truly like this soundtrack and I enjoy video...
4	2	If you've played the game, you know how divine...
5	2	I am quite sure any of you actually taking the...
6	1	This is a self-published book, and if you want...
7	2	I loved Whisper of the wicked saints. The stor...
8	2	I just finished reading Whisper of the Wicked ...
9	2	This was a easy to read book that made me want...

DATA CLEANING

Load the data into the Spark Data frame. Copied the data into a new data frame and checked the Null values in it. The dataset contains none of them. Used the drop_duplicates() function in the data frame to avoid ambiguity.



```
#creating a copy of the data
process_reviews=raw_reviews.copy()

#Checking for null values
missing_values = process_reviews.columns[process_reviews.isnull().any()]
print(missing_values)
```

```
Index([], dtype='object')
```

There are no null values.

```
#drop duplicates if any
process_reviews.drop_duplicates()
```

	Polarity	Text
0	2	This sound track was beautiful! It paints the ...
1	2	I'm reading a lot of reviews saying that this ...
2	2	This soundtrack is my favorite music of all ti...
3	2	I truly like this soundtrack and I enjoy video...
4	2	If you've played the game, you know how divine...
...	...	...
39995	1	The book boasts Intermediate - Advanced users,...
39996	2	I would highly recommend this product as it sa...
39997	2	I have an old, heavy duty garden hose that dev...
39998	2	I went way to long with a seized nozzle on my ...
39999	2	how can anyone add to the previous review? thi...

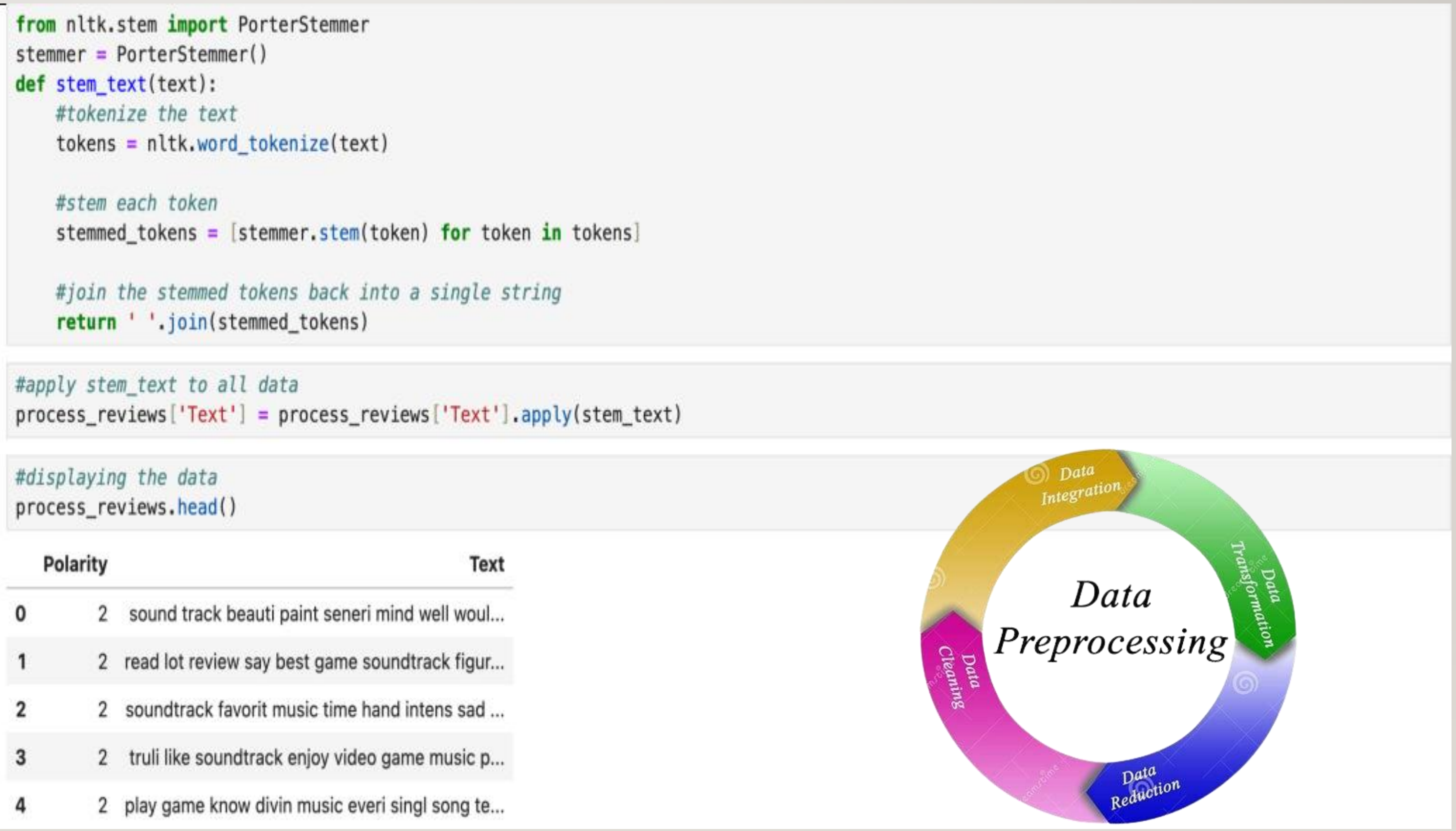
39983 rows × 2 columns

DATA PREPROCESSING

Using "re" package to remove unnecessary stop words including links, numbers, text in square brackets and punctuation.

By the preprocessed data we tokenized it, to get the list of words.

<u>PORTER STEMMER: </u>We use this text processing algorithm in NLTK to reduce words to their root form that enhances text analysis capabilities.

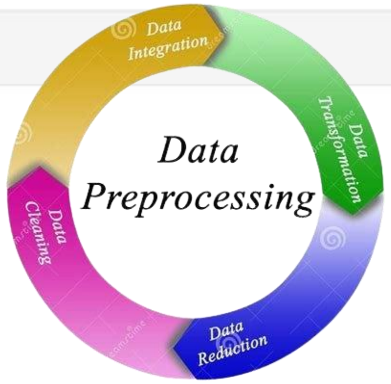


```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
def stem_text(text):
    #tokenize the text
    tokens = nltk.word_tokenize(text)

    #stem each token
    stemmed_tokens = [stemmer.stem(token) for token in tokens]

    #join the stemmed tokens back into a single string
    return ' '.join(stemmed_tokens)
```

```
#apply stem_text to all data
process_reviews['Text'] = process_reviews['Text'].apply(stem_text)
```

```
#displaying the data
process_reviews.head()
```

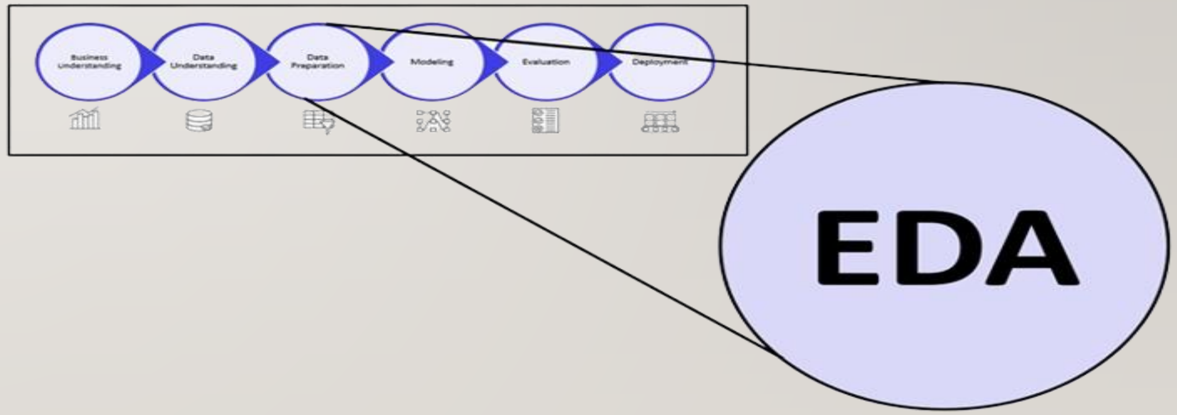
```
   Polarity                                               Text
0         2  sound track beauti paint seneri mind well woul...
1         2  read lot review say best game soundtrack figur...
2         2  soundtrack favorit music time hand intens sad ...
3         2   truli like soundtrack enjoy video game music p...
4         2  play game know divin music everi singl song te...
```



EXPLORATION DATA ANALYSIS (EDA)

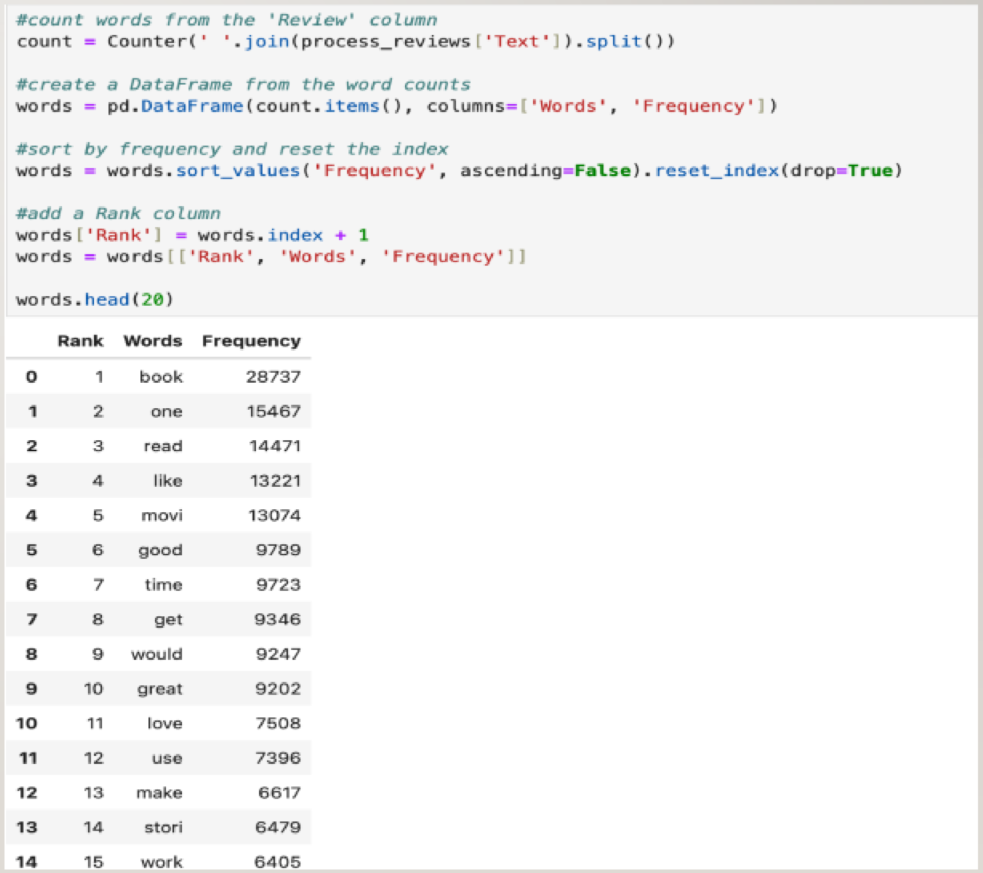
- Summarizing the main characteristics, patterns and trends in the dataset to gain insights and inform further analysis is done in the EDA process. This includes the following steps:

1. Data Loading
2. Data Exploration
3. Word Frequency Analysis
4. Data Cleaning and Pre Processing



EXPLORATION DATA ANALYSIS (EDA)

- 1. DATA LOADING: Uploading the text samples and sentiment labels (positive, negative, and neutral) which relates to the sentiment analysis dataset.
- 2. DATA EXPLORATION: To explore the length of the text samples to gain insights into the typical size of the dataset.
- 3. WORD FREQUENCY ANALYSIS: Using the "Review" column, we analyzed the frequency of words to identify positive, negative, and neutral sentiments in the dataset. Consider creating bar charts to visualize the most common words.
- 4. DATA CLEANING AND PREPROCESSING: Tend to any requirements related to data cleaning, including lowercasing text, managing missing values, and eliminating redundant characters.

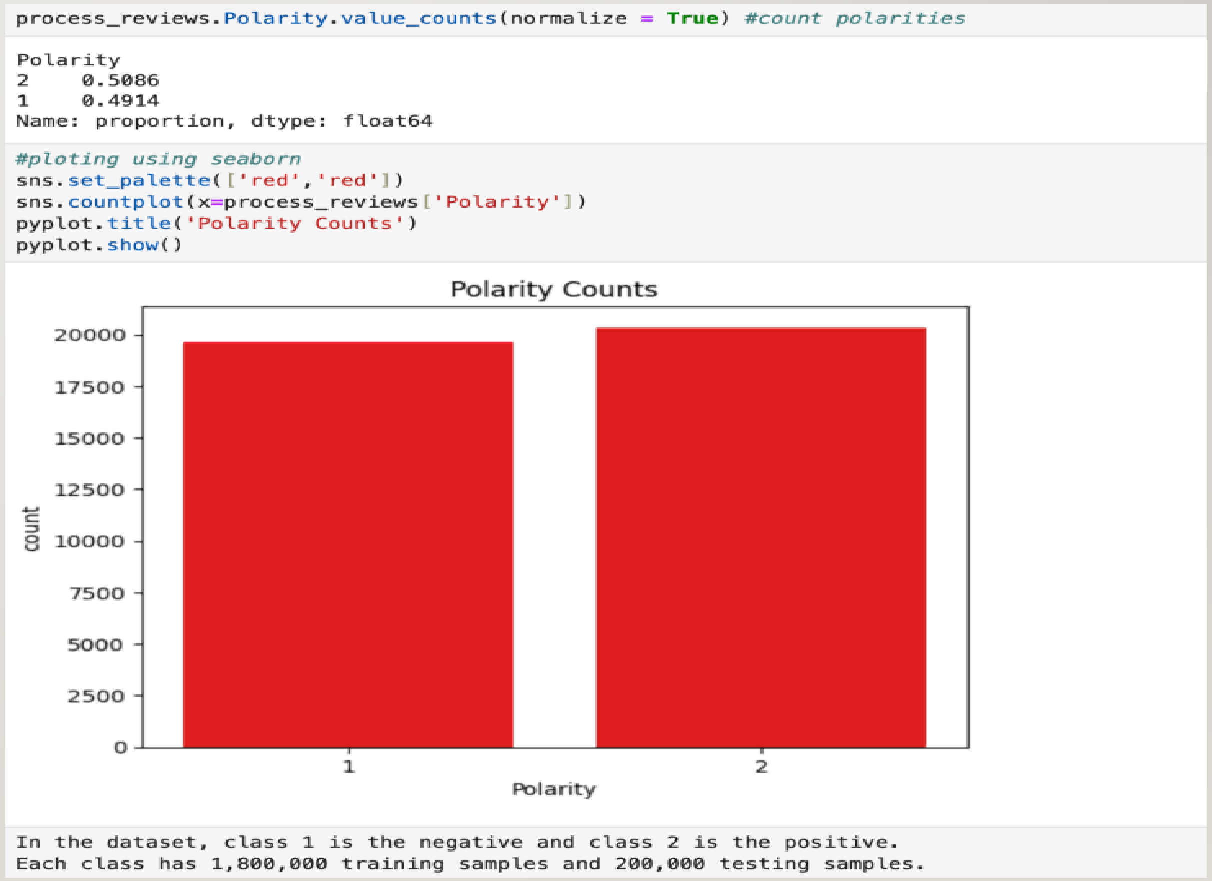


```
#count words from the 'Review' column
count = Counter(' '.join(process_reviews['Text']).split())

#create a DataFrame from the word counts
words = pd.DataFrame(count.items(), columns=['Words', 'Frequency'])

#sort by frequency and reset the index
words = words.sort_values('Frequency', ascending=False).reset_index(drop=True)

#add a Rank column
words['Rank'] = words.index + 1
words = words[['Rank', 'Words', 'Frequency']]

words.head(20)
```

	Rank	Words	Frequency
0	1	book	28737
1	2	one	15467
2	3	read	14471
3	4	like	13221
4	5	movi	13074
5	6	good	9789
6	7	time	9723
7	8	get	9346
8	9	would	9247
9	10	great	9202
10	11	love	7508
11	12	use	7396
12	13	make	6617
13	14	stori	6479
14	15	work	6405

CHALLENGES

- Here, we visualized the distribution of polarity values in the 'Polarity' column of the process_reviews Data Frame.
- Unfortunately, we faced a difficulty over here as we couldn't define the difference between positive and negative emotions much. We concluded that the accuracy is not up to the mark through the 'Polarity' column in the dataset.



```
process_reviews.Polarity.value_counts(normalize = True) #count polarities
```

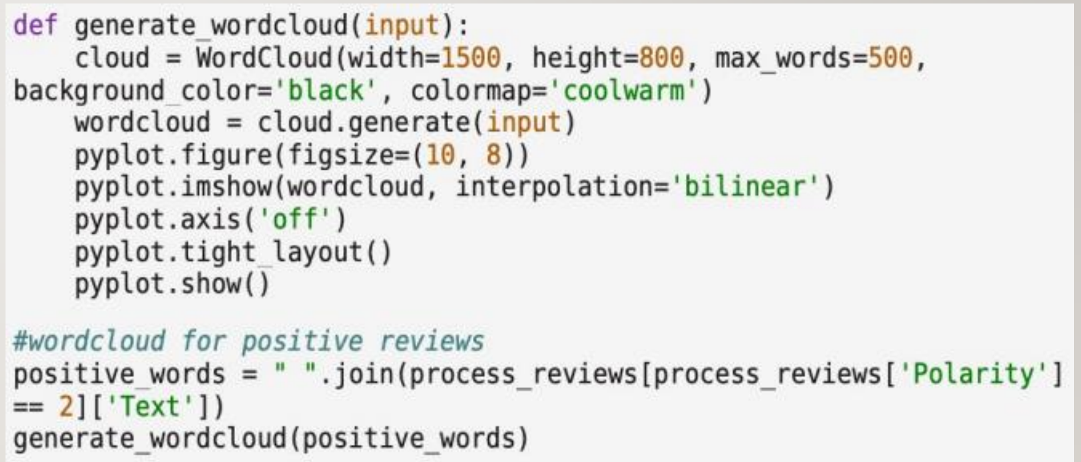
```
Polarity
2    0.5086
1    0.4914
Name: proportion, dtype: float64
```

```
#ploting using seaborn
sns.set_palette(['red','red'])
sns.countplot(x=process_reviews['Polarity'])
pyplot.title('Polarity Counts')
pyplot.show()
```

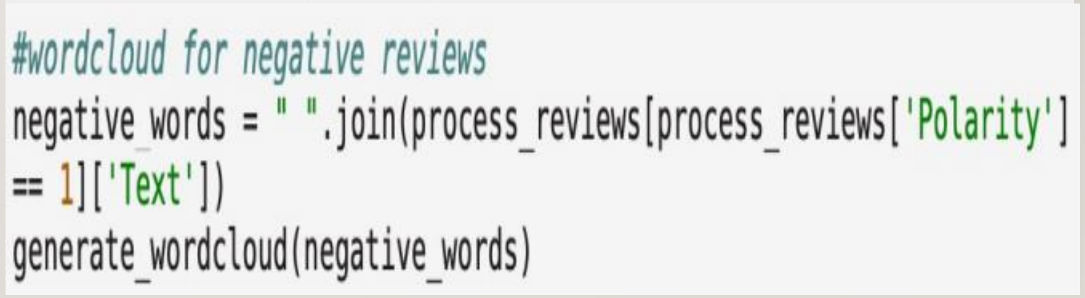
```
In the dataset, class 1 is the negative and class 2 is the positive.
Each class has 1,800,000 training samples and 200,000 testing samples.
```

VISUALIZATION

Generate_wordcloud() is a Python function, which generates and shows a word cloud representation. We used WordCloud library for creating word clouds, and Matplotlib to present the visualization. The result is a word cloud that graphically displays the terms that appear most frequently in the positive ratings, providing a summary of the key concepts related with the positive sentiments in the dataset. We created a horizontal bar plot, To display the top 25 'Positive' words and their occurrences in the input text.

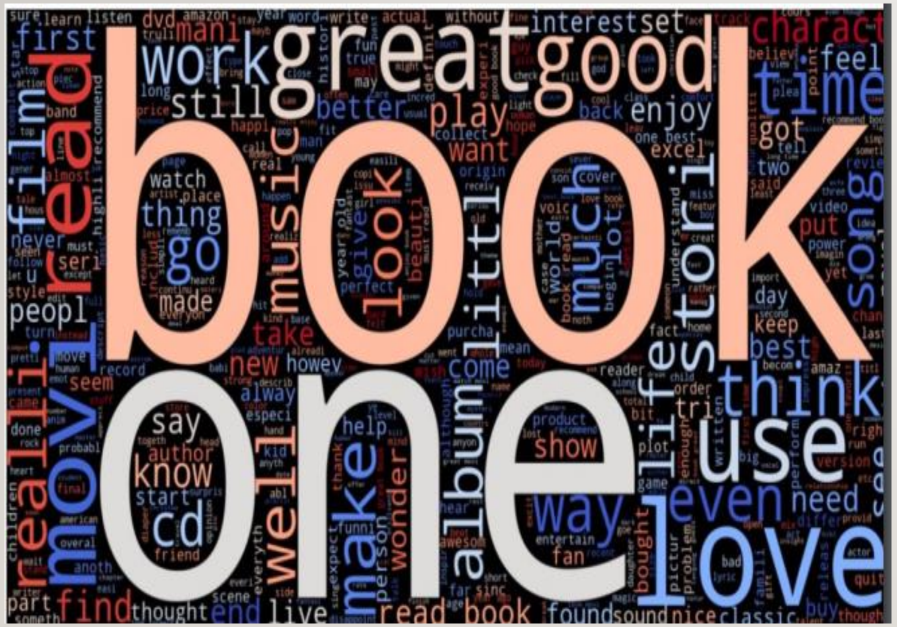


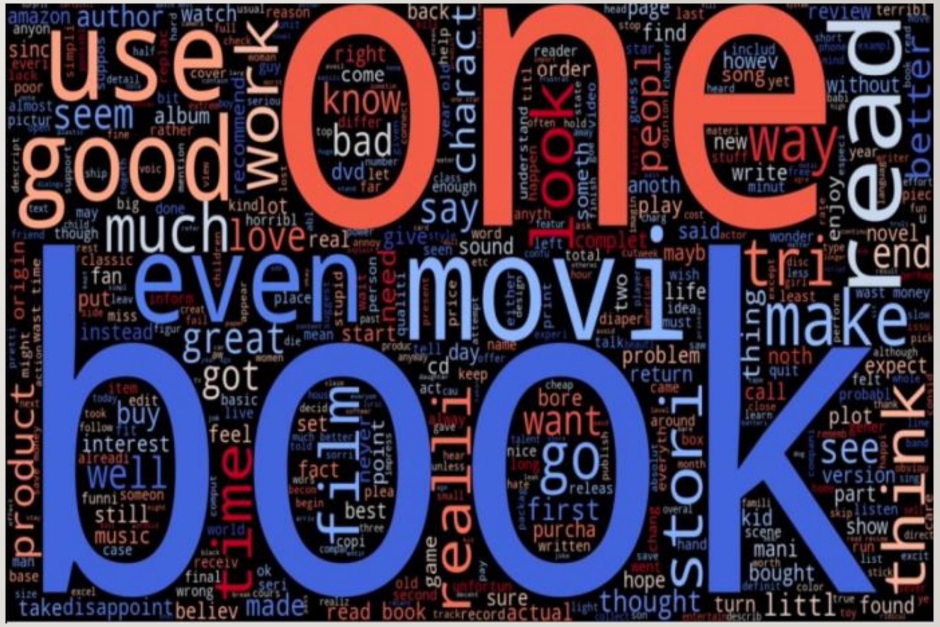
```python
def generate_wordcloud(input):
    cloud = WordCloud(width=1500, height=800, max_words=500,
background_color='black', colormap='coolwarm')
    wordcloud = cloud.generate(input)
    pyplot.figure(figsize=(10, 8))
    pyplot.imshow(wordcloud, interpolation='bilinear')
    pyplot.axis('off')
    pyplot.tight_layout()
    pyplot.show()

#wordcloud for positive reviews
positive_words = " ".join(process_reviews[process_reviews['Polarity']
== 2]['Text'])
generate_wordcloud(positive_words)
```



```python
#wordcloud for negative reviews
negative_words = " ".join(process_reviews[process_reviews['Polarity']
== 1]['Text'])
generate_wordcloud(negative_words)
```

OUTPUT





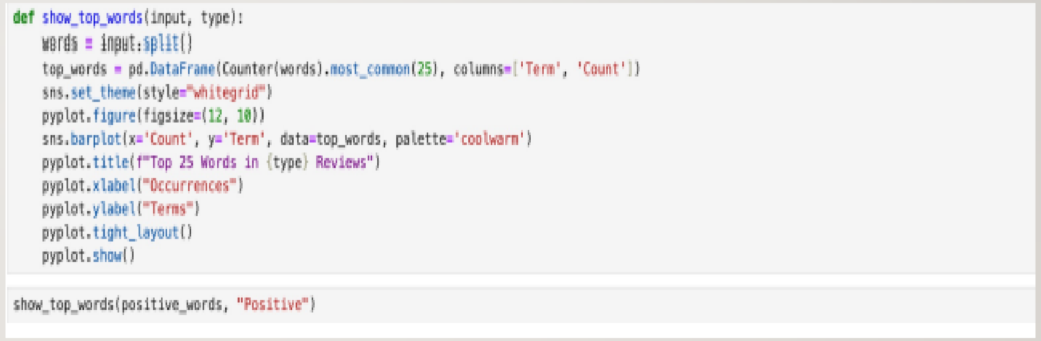
Positive reviews output

Negative reviews output

As we can see these words are reoccurred in most of the reviews and tended as the negative one's.

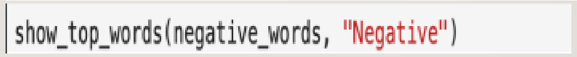
BAR PLOT REPRESENTATION

- We created a horizontal bar plot, To display the top 25 'Positive' words and their occurrences in the input text. Splitting Input Text: split() method, divides the input text into a list of words. To find the number of times each word appears in the list, we used the Counter class from the collection's module. The top 25 most common words and their respective counts are then obtained by using the most_common(25) method. OUTPUT CONCLUSION: We can say that these are the top 25 positive words that are occurred highest number of times in the customer reviews. By the extraction of these words, we can draw the conclusion that if the frequency of these words are higher it leads a path to the positive sentimental analysis.



```
def show_top_words(input, type):
    words = input.split()
    top_words = pd.DataFrame(Counter(words).most_common(25), columns=['Term', 'Count'])
    sns.set_theme(style="whitegrid")
    pyplot.figure(figsize=(12, 10))
    sns.barplot(x='Count', y='Term', data=top_words, palette='coolwarm')
    pyplot.title(f"Top 25 Words in {type} Reviews")
    pyplot.xlabel("Occurrences")
    pyplot.ylabel("Terms")
    pyplot.tight_layout()
    pyplot.show()
```

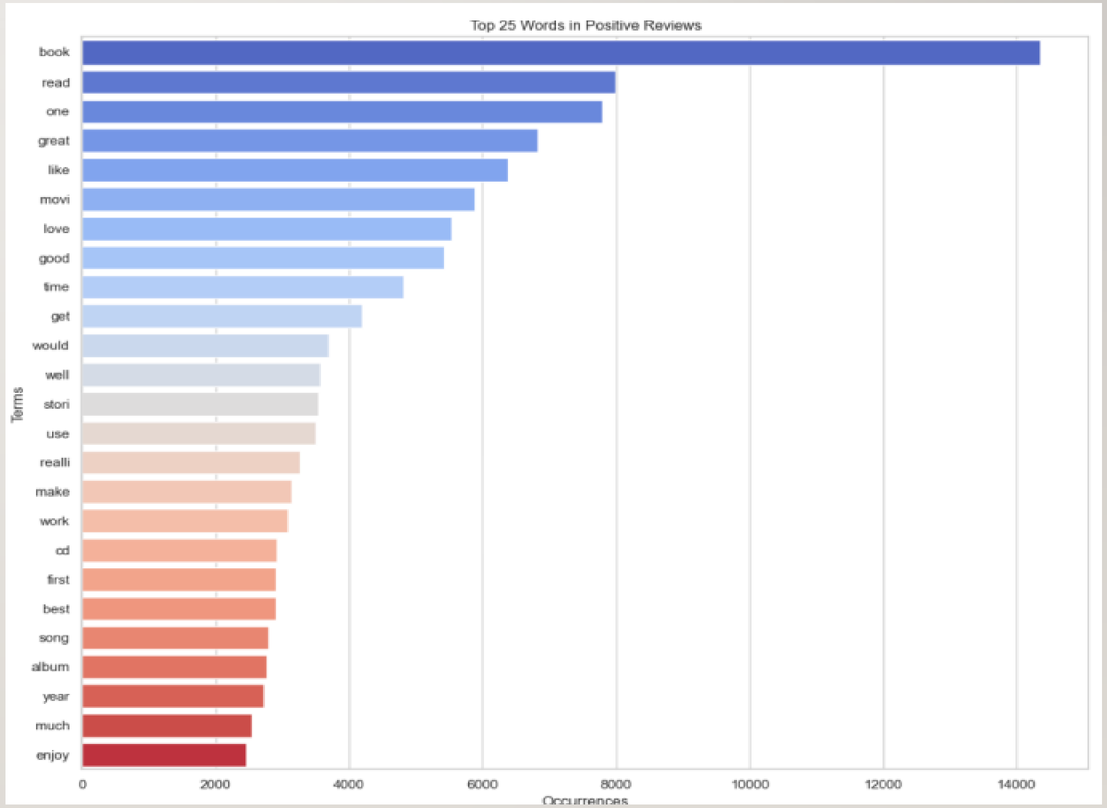
```
show_top_words(positive_words, "Positive")
```



```
show_top_words(negative_words, "Negative")
```

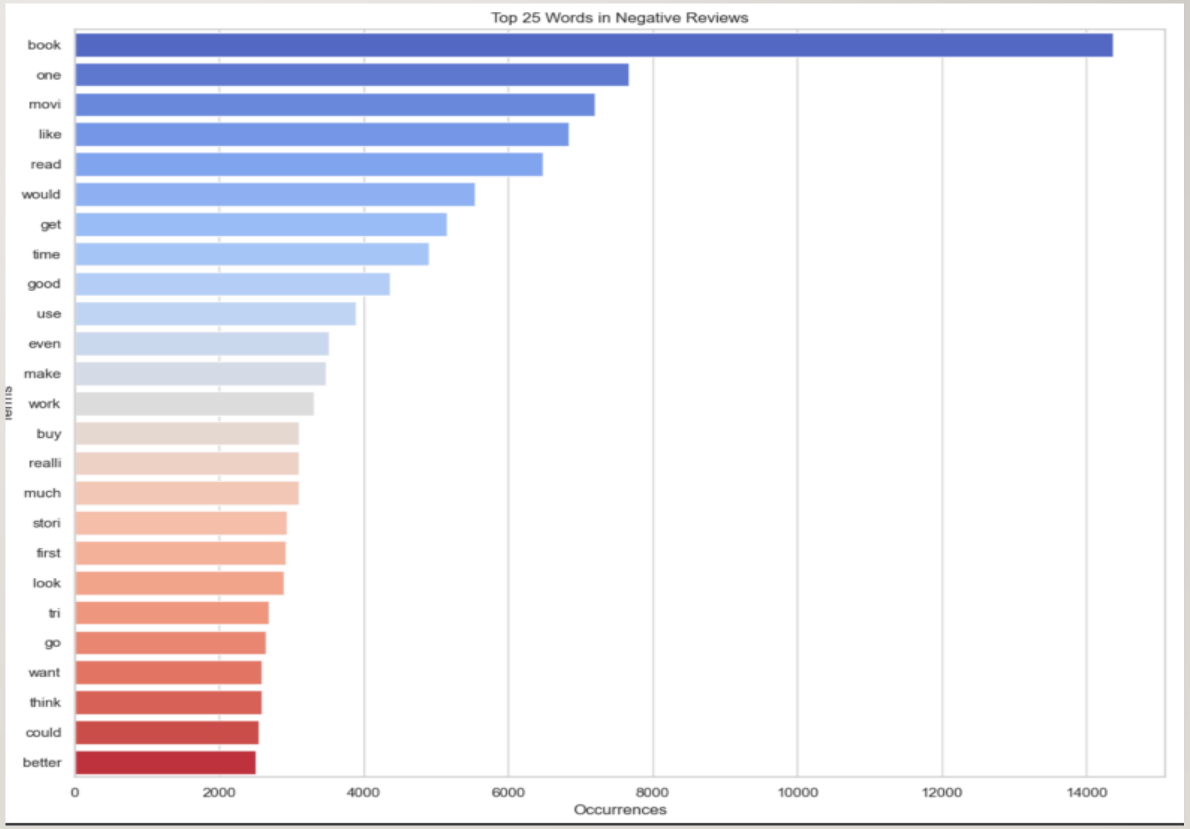
POSITIVE OUTPUT CONCLUSION:

We can say that these are the top 25 positive words that are occurred highest number of times in the customer reviews. By the extraction of these words, we can draw the conclusion that if the frequency of these words are higher it leads a path to the positive sentimental analysis.



NEGATIVE OUTPUT CONCLUSION:

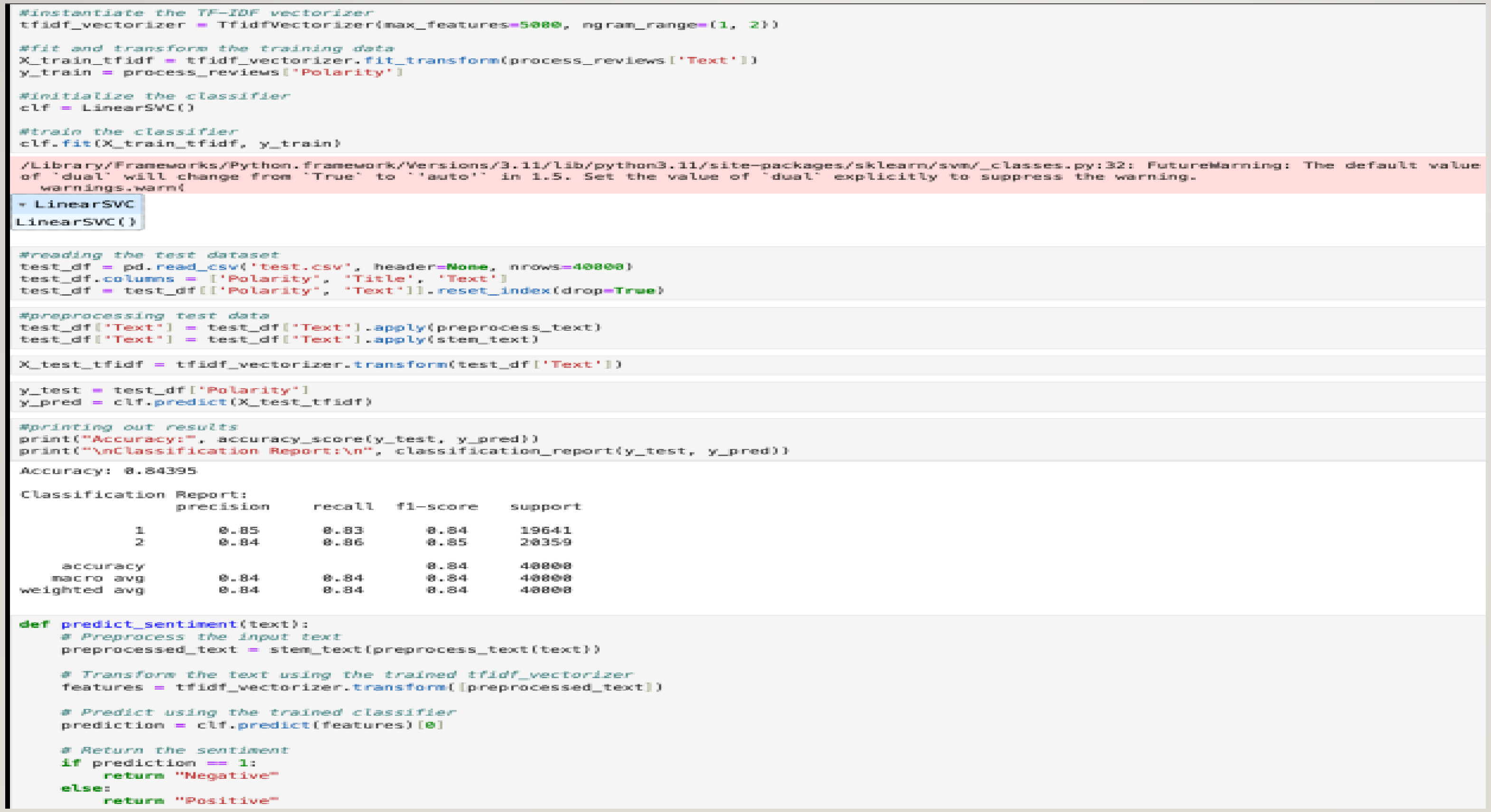
Based on the frequency of occurrence, we can identify the top 25 negative terms in customer reviews. Through the process of word extraction, we can conclude that a higher frequency of these words indicates a greater possibility of negative sentimental analysis.



TRAINING MODEL

Here we are using TfidfVectorizer class for sentimental analysis. max_features parameter is set up to 5000 which will take top words or combinations for classification. The ngram_range parameter is set to (1, 2), indicating that unigrams and bigrams will be considered. LinearSVC is designed for binary classification tasks, meaning it classifies instances into one of two classes. Created an instance from the LinearSVC class and then trained the LinearSVC model, making predictions, and evaluating the accuracy of the model. accuracy_score (y_test, y_pred): This function finds the model's accuracy by comparing the true labels (y_test) with the predicted labels (y_pred). Overall, there is an accuracy score achieved is 0.84395 and as per the accuracy matrix we can conclude that the average accuracy obtained is 84%.

QUERY FOR TRAINING MODEL



```
#instantiate the TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))

#fit and transform the training data
X_train_tfidf = tfidf_vectorizer.fit_transform(process_reviews['Text'])
y_train = process_reviews['Polarity']

#initialize the classifier
clf = LinearSVC()

#train the classifier
clf.fit(X_train_tfidf, y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/svm/_classes.py:32: FutureWarning: The default value
of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
▾ LinearSVC
LinearSVC()
```

```
#reading the test dataset
test_df = pd.read_csv('test.csv', header=None, nrows=40000)
test_df.columns = ['Polarity', 'Title', 'Text']
test_df = test_df[['Polarity', 'Text']].reset_index(drop=True)
```

```
#preprocessing test data
test_df['Text'] = test_df['Text'].apply(preprocess_text)
test_df['Text'] = test_df['Text'].apply(stem_text)
```

```
X_test_tfidf = tfidf_vectorizer.transform(test_df['Text'])
```

```
y_test = test_df['Polarity']
y_pred = clf.predict(X_test_tfidf)
```

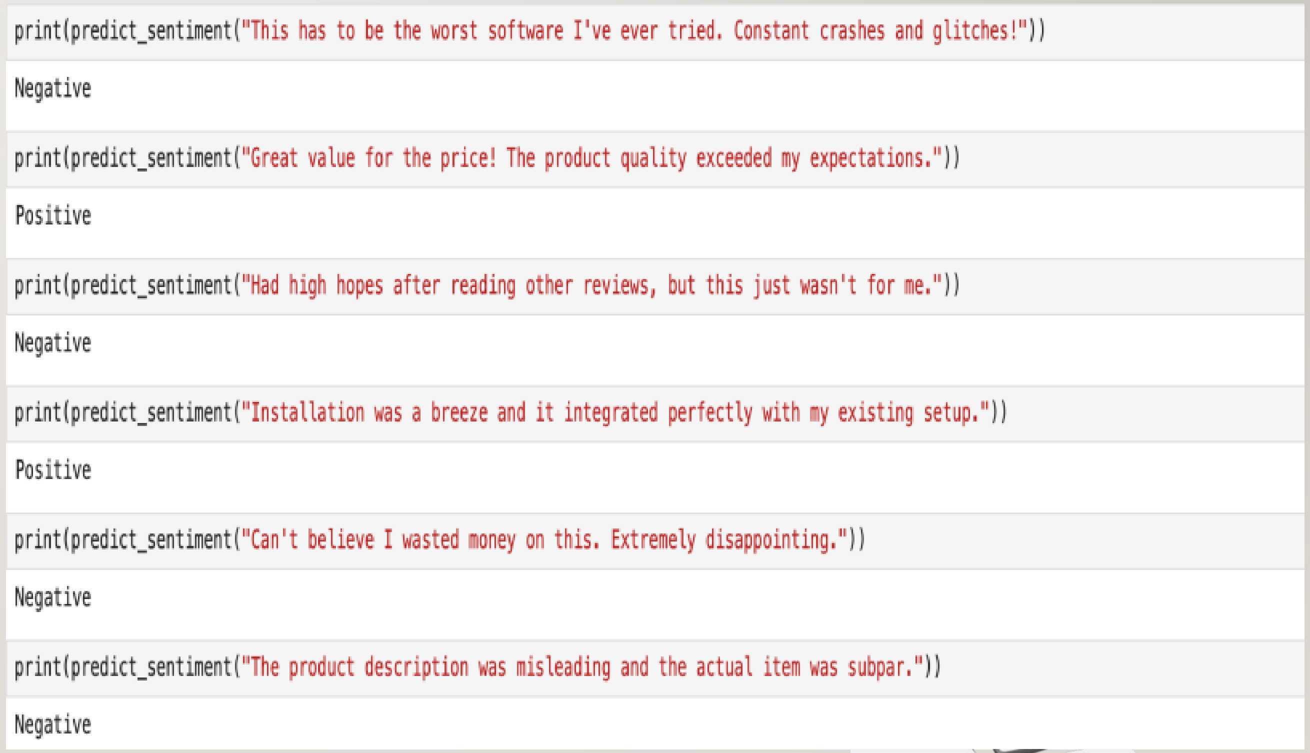
```
#printing out results
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.84395

Classification Report:
               precision    recall  f1-score   support

           1       0.85      0.83      0.84     19641
           2       0.84      0.86      0.85     20359

    accuracy                           0.84     40000
   macro avg       0.84      0.84      0.84     40000
weighted avg       0.84      0.84      0.84     40000
```

```
def predict_sentiment(text):
    # Preprocess the input text
    preprocessed_text = stem_text(preprocess_text(text))

    # Transform the text using the trained tfidf_vectorizer
    features = tfidf_vectorizer.transform([preprocessed_text])

    # Predict using the trained classifier
    prediction = clf.predict(features)[0]

    # Return the sentiment
    if prediction == 1:
        return "Negative"
    else:
        return "Positive"
```

TESTING

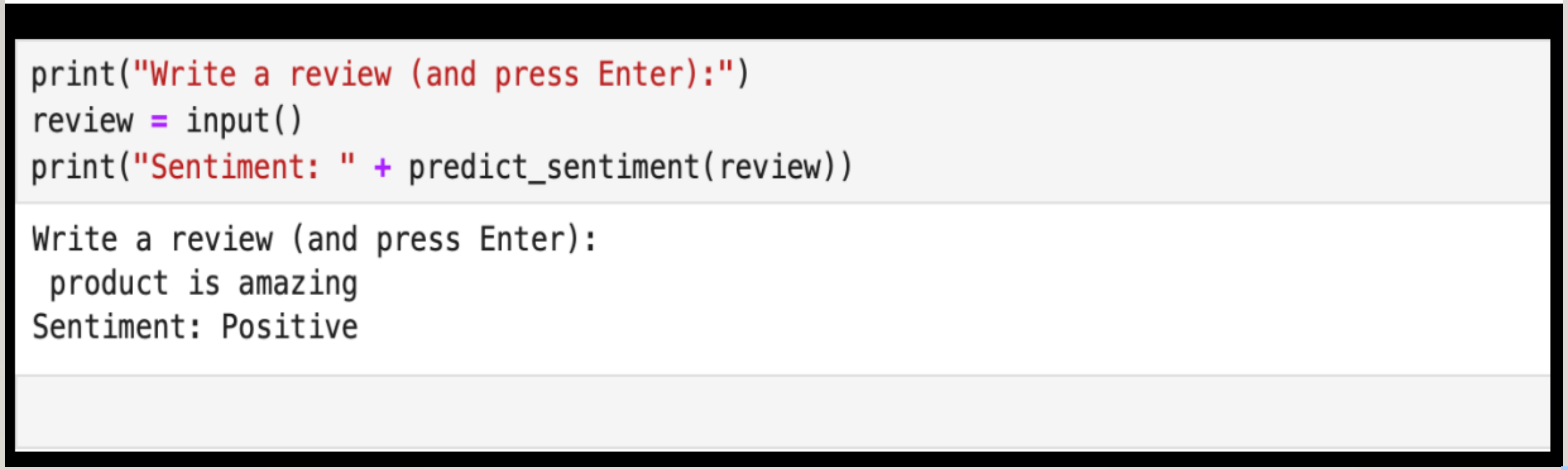
Here, we tested the dataset by using the sample reviews. The 'predict_sentiment' function is a part of sentimental analysis model, that has been trained to predict sentiment labels from input text. In this case the function takes the input from the user, preprocess it if necessary and gives the sentimental analysis of the input text that is either positive, negative or neutral.



```
print(predict_sentiment("This has to be the worst software I've ever tried. Constant crashes and glitches!"))
Negative
print(predict_sentiment("Great value for the price! The product quality exceeded my expectations."))
Positive
print(predict_sentiment("Had high hopes after reading other reviews, but this just wasn't for me."))
Negative
print(predict_sentiment("Installation was a breeze and it integrated perfectly with my existing setup."))
Positive
print(predict_sentiment("Can't believe I wasted money on this. Extremely disappointing."))
Negative
print(predict_sentiment("The product description was misleading and the actual item was subpar."))
Negative
```

USER END TESTING

At last, we input a review, soon after it predicts the sentiment polarity of the entered review using the predict_sentiment function.



```
print("Write a review (and press Enter):")
review = input()
print("Sentiment: " + predict_sentiment(review))
```

```
Write a review (and press Enter):
 product is amazing
Sentiment: Positive
```

Thank you!