Syed Muhammad Shahryar Zaidi
Assessment 3: Web Crawler and NLP Systems

Overview:

The Issue:

In this contemporary world with ever changing technological advancements reviews form a bases for the customers for helping in making a purchase decision. From things as small as a lug nut to as big as a car people daily visit websites and in particular their product review sections in order to see what the prior owners or users of their intended to be purchased product have to say about it. Upgrading your system means a decision that will require a lot of research for compatibility, effectiveness and efficiency. At the same time, it is also an expansive decision to make so a thorough research in that regards is key to making a prolific decision, that can make sure that you are able to suffice the requirements to run the large software of your liking. (Grand View Research, 2018)

In this project we will analyse one of the **biggest E-commerce websites Amazon.in with domain location in India for customer reviews and rating regarding one of the computers operational product solid state drive in order to see what are consumers reactions after buying that product that can be analysed to assist future buyers as well as the company that is selling them for feedback and service and product upgradation.**

In this analysis our major focus is not just to assess the basic reviews based on NLP techniques but to build a foundational structure where any other product, product category or brand can be scrutinized and both the merchant and the consumers can use this mechanism to assess the popularity of their product from the merchants point of view and the performance of the product from the consumers point of view as this is the very essence of what data science and its methodologies are there for.

Issue in relation to Natural Language and its Meta Data:

The reason of selecting the reviews is that it is all based on words and reactions of the people consuming the product. The data that is acquired has character data as well as numeric data exported in csv format in the due process for processing that consists of review headings, rating (1-5 star) and review body with peoples reactions to it.

Presence of the Issue on the world wide web:

There are multiple domains that host E-commerce websites with a lot of product offerings of similar sort. There are websites that offer multi products and websites that are associated to one product only. So, choosing between them to search for your product is difficult and to select from which one it is to be bought is even difficult, so, narrowing it down to one website is what is being done in this project.

Web Crawler Alignment:

Presence, Relatability/Relevance and Expandability:

In search of the relevant information and available data on the issue at hand there were multiple domains that offered the relevant data and information to be crawled. Some prominent ones that were looked upon were Amazon.in, Samsung.com.in and flipcart.com. The search was narrowed down to Amazon.in as the review section that was the initial point of consideration was exhaustive on the product from the brand Samsung.

The reason for selection is that the domain and in particular the web page offered an exhaustive data on the SSD drives offered by Samsung that have multiple compatibility options available and are easy to use and install compared to all the other brands as quoted by the customers in their reviews that we will see later in this report.

The main variables that are consumed from the domain for the Natural language processing are star ratings (a quantifiable value), headers of the comments (an initial point of initiation) and the review body (a description of the emotions of the consumers). These all considerations have been made keeping in mind the issue at hand where our concern is to see what the users of an SSD offered by Samsung has to say and is it a worth buy or not.

Web crawling request package is a powerful approach for gathering information from the internet by locating all the URLs for one or more domains. In our case there were a total of 51 URLs of Amazon.in that were crawled in order to scrape the data and so a manual imputation of such magnitude for generation of data sets would be a cumbersome and tiring job so in order to minimize the work this tool is brough in use. The easy-to-use nature of this package and easy understandability of this package for any everyday developer will make it easy for anyone to take on this pilot crawler and build on it in order to be able to do further alterations and do more intense relevant URLs collections on different domains and projects of similar kind for any other product or product subcategory.

NLP Task Alignment:

With words being the core aspect of our research, we will be using Sentiment analysis and the Latent Semantic Indexing. The reasons behind using these two tasks where the sentiment analysis is used because in our case our focus is to analyse how people are reacting to the product in question after using it and for any company and as well as a prospective buyer feedback plays an important role and to analyse each comment separately is a time consuming and cumbersome process. Moreover, if we are looking from a company perspective Sentiment analysis is essential to detect and understand customer feelings. Sentiment analyses generate insights into how companies can enhance the customer experience and improve customer service. For the reason of the usage of Semantic indexing examines the abstract structure of phrases in order to discover the most important aspects in a text and comprehend the subject. It also identifies the connections between the document's numerous concepts. In our case it also helps in discovering the relevance of colloquial phrase in the review section. Furthermore, it helps find an answer to a question without having to ask any of the users what is going on in their mind and what was their topic of focus when they wrote that comment/feedback. Lastly, it is also the means of extraction of relevant and useable information from large amounts of unstructured data.

In feedback analysis the main purpose is to see what the consumers are thinking about the product and to analyse any review section of any given product the best way to do that is to look for recognizable text to a given emotion for which Semantic indexing is used and to see how many positive and negative reviews there for which Sentiment analysis is used.

Furthermore, the WebCrawler brings in URLs from the Amazon.in 51 pages containing reviews and star ratings that will help in passing in data into NLP task where the star rating provides data combined with the reviews becomes input for the sentiment analysis and the body text becomes input for the Semantic indexing.

Therefore, for the NLP techniques to be implemented we have first we clear the sentences from all the punctuations, emojis' and two words that have low or no implementation on the input data. Finally, Stemming and lemmatization is carried out on the corpus data. There are operations carried out to make N-grams, Bi-grams and Trigrams with lastly carrying out WordCloud before passing the data into NLP models of Sentiment and Semantic.

Web Crawler Domain Discussion:

With the WebCrawler being built the main question that arose was to choose which domain to crawl in order to scrape the data/corpus that will make the foundation of our whole analysis. So, after careful assessment of multiple domains that offered the data on the question at hand Amazon.in was selected from Flipcart, Samsung.in and couple of other E-commerce websites.

Amazon.in offered an exhaustive number of URLs to be crawled and offered a wide range of data on the reviews and feedback on user experience that was our initial point of consideration for this whole project. A total of 256 URLs were available to be crawled full of reviews and feedbacks from the consumers and so the crawler was limited to a total of 51 crawled URLs in order to keep the operations going not facing any hinderance from the server site regarding scrape bot prohibition.

In relation of the coverage of the data on the domain on our issue at hand the domain offered multiple brands offering with sub-categories of different sizes of storage capacity offerings and multiple offerings of sub-primary brands with a certain brand. Looking at such a huge variety of astronomical data of raw corpus sufficed the requirement of the analysis of offering complete coverage on the issue making sure that sufficient reviews can be scraped in order to drive meaningful insights from the scraped data when it will be processed using the NLP techniques.

At the initial point of interaction with the URLs the Natural language data that was on offer was huge in size where there was item description with item name, price tags, star ratings and reviews with reviews title. For the sake of keeping the assessment efficient three components from the crawled websites were picked up consisting of the: Star ratings, Title of reviews and the Review body itself. These consists of a numeric variable of star ratings and two string variables of title and the review. These all three variables prove to be sufficient for analysis and model building purposes as they cover the core aspect of our real issue being investigated i.e., insight on consumer feedback on a certain product to see the popularity of that product and customer satisfaction levels.

The website disallows spider bot and many other bots but if the user agent is set to four different web browsers access paths as shown in the code chunk below (Code Chunk 1) then the issue is resolved with very less restrictions for crawling and scraping using the robots.txt checking mechanism.

Code Chunk 1:

###Links and Headers

HEADERS = ({'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36',

'Accept-Language': 'en-US, en;q=0.5'})

Web Crawler Workflow:

Now moving onto the original web-crawler that was designed we designed that basic technology components that were used were importing the request package with beautiful soup. The initial consideration for using the request package in comparison to scrapy and selenium was that for scrapy the spider bot that it uses to crawl that URLs was denied access when searched and found in the robots.txt file. For Selenium the aftermath of crawling the scraping process was turning out to be too slow in comparison to request package and the browser that was created was generating a bigger network traffic.(Igor Savinkin, 2020) In case of request package, the coding was pretty routine and the web traffic that was generated was smaller in comparison to selenium and with using the user agent with different browsers IP's the restriction elements by the domain itself reduced in number and access was granted seamlessly as in the code chunk 1.

Moving on, the main challenge faced was in initial browsing where selecting the URL that complies with the initial issue was very difficult to find that had ample data to comply with the data corpus requirements. But once that domain and URLs were identified the target data was found in the product review section of the product under review. It had 256 webpages but that was limited to 51 to suffice the corpus needs as well as keep the data workable. There is a notable element in the whole coding that was prepared that the whole web crawling code was made fault tolerant where a IF, ENDIF loop was used to fetch the URLs with +1 increase breaking the loop till the limit of page not found as shown in the Code Chunk 2 below:

Code Chunk 2:

```
def main():
    baseUrl = "https://www.amazon.in"
    mainCategory = "electronics"
    productCategory = "Samsung SSD"
    pagesToFetch = 51
    productObjectDataset = []

    print("Processing...")
    ## iterate over amazon pages where upper limit is a big number as we donts know how many pages there can be
    for i in range(1, pagesToFetch + 1):
        urlToFetch = baseUrl + "/s?k=" + productCategory + "&i=" + mainCategory
        if (i > 1):
            urlToFetch += "&page=" + str(i)
        #endif
```

```
res = requests.get(urlToFetch)


soup = BeautifulSoup(res.text, 'html.parser')
content = soup.find_all('a',
                        class_='a-link-normal a-text-normal',
                        href=True)


print("Fetching: " + urlToFetch)


# breaking the loop if page not found
if (len(content) == 0):
   print("Nothing found in: " + str(i))
   break
#endif
```

Sequencing of the Crawler:

As the crawler was being developed using the intended packages the first thing that was done was based on defining the base URL with that was to be crawled and what classes to be crawled with importance given to main category that is electronics and the product category defined as the Samsung SSD and the page to be fetched limited to 51 for the reasons already discussed above.

The single URL was defined as shown in Code Chunk 3 where different pages of product review were crawled in an iteration manner till the loop that was used for fault tolerance was broken with no page found. There is a sort of a bias or kind of limitation of the project that only a single URL was defined for the crawler with multiple web pages to be crawled for the product review.

Code Chunk 3:

```
### Link to the amazon product reviews
url = 'https://www.amazon.in/Samsung-Internal-Solid-State-MZ-V7S500BW/product-reviews/B07MFBLN7K/ref=cm_cr_arp_d_paging_btm_next_2?ie=UTF8&reviewerType=all_reviews&pageNumber='
review_list = []
```

Then the beautiful soup package was used to scrape the attained URLs that were saved in a json product review file and the review titles, review star ratings and the review body was scraped to be formed into a combined file named as review_list that contained all the reviews with its title, ratings

and body matter that will be later processed using the NLP and machine learning techniques. Lastly, the whole file was converted to a python data frame and later was converted into a excel file that would be further used down the process. This whole process is showed in the Code Chunk 4 below:

Code Chunk 4:

```
review_list = []


def retrieve_reviews(soup):
    # Get only those divs from the website which have a property data-hook and its value is review
    reviews = soup.find_all("div", {'data-hook': "review"})


    # Retrieving through the raw text inside the reviews
    for item in reviews:


        review = {
        # Get the title of the review
        'title': item.find("a", {'data-hook': "review-title"}).text.strip(),


        # Get the rating. It will be like 4.5 out of 5 stars. So we have to remove out of 5 stars from it and only keep float value 4.5, 3.4, etc.
        'rating': item.find("i", {'data-hook': "review-star-rating"}).text.replace("out of 5 stars", "").strip(),


        # Get the actual review text
        'review_text': item.find("span", {'data-hook': "review-body"}).text.strip()
        }


        review_list.append(review)


# Get the page content from amazon
# as we know we have 43 pages to visit and get content from
for pageNumber in range(1, 51):
    raw_text = requests.get(url=url+(str(pageNumber)), headers = HEADERS)
```

```
    soup = BeautifulSoup(raw_text.text, 'lxml')
    retrieve_reviews(soup)

for index in range(len(review_list)):
    # Print out all the reviews inside of a reviews_list
    print(f"{index+1})  {review_list[index]}")
    print("")

import csv
import pandas as pd

# Create dataframe out of all the reviews from amazon
reviews_df = pd.DataFrame(review_list)

# Put that dataframe into an excel file
reviews_df.to_excel('samsung.xlsx', index = False)

print("Done.")
```

As discussed in the ongoing section the first crawled URLs were stored in a json file and later that json file was converted into a review list in python that was later converted into a data frame that later was finally saved as a excel file as shown in the last parts of the code chunk 4.

EDA of the harvested Corpus:

This section describes how our generated corpus looks like after all the web crawling and the scraping on those attained URLs have been carried out. The generated corpus has 320 entries with three columns to it as described in the data fetching section and shown in the Code Chunk 5 and output 1 below:

Code Chunk 5:

```
reviews_df.shape
```

Output 1:

```
(320, 3)
```

The initial data frame that is generated has raw form of data to it so in order to make it into a usable data so there a multiple cleaning process that are carried out in order to make it into a more viable form of input. Firstly, there are some emojis that are found in the corpus that need to be removed. So, the emoji package offered by python is used to remove the unnecessary elements in the data set as shown in the Code Chunk 6. Furthermore, a cleaning process is carried out where all the punctuations from the review titles and the body are being removed with further processes involving turning all the capital letters to small letters and removing all the @ mentions from all the reviews.

Code Chunk 6:

```
import emoji


def remove_emojis(text):
    reg = emoji.get_emoji_regexp()
    emoji_free_text = reg.sub(r'', text)
    return emoji_free_text
```

Then, the words are tokenized and words that are less than two letters in total are removed to make sure that they don't impact the outcomes of the ML techniques that are to be used later in the project. A later assessment of the cleaned data frame showed that there are many unnecessary 3 letter words that are to be found in the corpus so a further 3 letter word cleaning was carried out in order to make the text data easy to process and used for analysis as shown in Code Chunk 7.

Code Chunk 7:

```
# Cleaining function
def preprocess(reviews, stopwords):
    cleaned_reviews = []
    for review in reviews:
        lower_text = review.lower()

        punctuations = '''`!()-[]{};:'"\,<>./?@#$%^&*_~=+°'''

        lower_text = re.sub(r"@[A-Za-z0-9]+", "", lower_text)   # Removes the @mentions from the tweets
        lower_text = re.sub(r"[0-9]+", "", lower_text)        # Removes the Numbers from the tweets

        # tokenization
```

```
        tokens = word_tokenize(lower_text)

        # Removing stopwords
        filtered_text = [word for word in tokens if word not in stopwords]

        # look for empty words or words just made of two letters and remove that
        for token in filtered_text:
            if token == "":
                filtered_text.remove(token)

        filtered_text = ' '.join([word for word in filtered_text])

        clean_text = remove_emojis(filtered_text)

        # Removing punctuations in string
        # Using loop + punctuation string
        for ele in clean_text:
            if ele in punctuations:
                clean_text = clean_text.replace(ele, "")

        # Removing small words with length less than 3
        clean_text = ' '.join([t for t in clean_text.split() if len(t)>=3])

        cleaned_reviews.append(clean_text)

    return cleaned_reviews
```

After all these pre-process steps carried out the size of the corpus remained the same but the sentences that remained were ready to be processed as an input in the next step. The summary of the corpus can be seen in the output 2 below:

Output 2:

```
['brand brand always read write speed ssd flawless successfully installed
macbook pro additional adapter aliexpress guys remember samsung evo plus
compatible macbook tried crucial wasted tried closing laptop lid hours get
hibernate shutdown issues one thing anyone upgrading like make sure use
capton high heat resistant tape instead tapes ssd may lead ssd processor
damage thanks seller amazon safe fast delivery',
 'lot research comparison ssds decided give try swear decision right makes
huge difference literally feel like flying even though using lenovo
processor ramit read write speed incomparable read around mbps write
mbpsboot speed quadrupled programs ready gobattery life laptop increased
even disconnected sata hdda worthy upgrade worth regret even buy',
 'acer nitro detect first boot installing secondary pcie express slot
scared messed checked bios find drive got detected went disk management
initialized gpt type allocated drive tasted superfast speed nvme hate
samsung liked drive much wold recommend people want see system responding
ever lightning speed lap purchased hynix could offer half speed
samsungnitro actually works hdd time also replaced rpm hdd ssd',
```

In order to have an initial idea about what the data frame offered to work with some initial visualizations were also carried out in form of a bar graph as shown in figure 1 and a pie chart as shown in figure 2 with their Code Chunk 8 and 9 that produced them as below:

Code Chunk 8:

```
plt.figure(figsize = (7, 7))
sns.countplot(reviews["rating"])
```

Code Chunk 9:

```
explode = [0.05, 0.04, 0, 0.02, 0]

names = ["Rating 5.0", "Rating 4.0", "Rating 1.0", "Rating 3.0", "Rating 2.0"]

plt.figure(figsize = (10, 10))

plt.pie(rating_count["rating"],

      labels = names,

      labeldistance=1.05,

      wedgeprops = { 'linewidth' : 1.5, 'edgecolor' : 'white' },

      explode = explode,

      autopct = '%.2f%%',

      shadow = True,

      pctdistance = .85,

      textprops = {"fontsize": 14, "color":'w'},

      rotatelabels = True,

      radius = 1.3

     )

plt.show()
```

Figure 1:

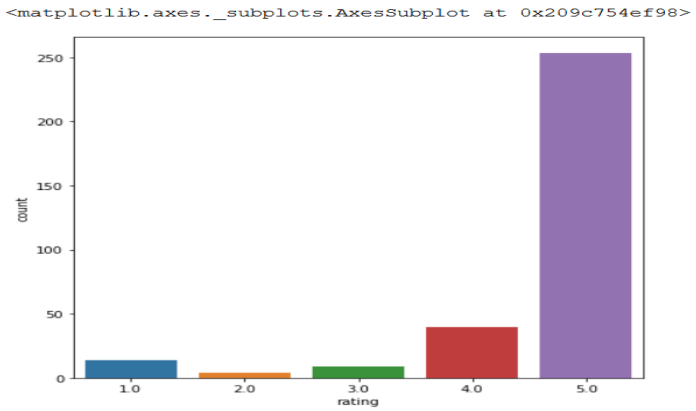
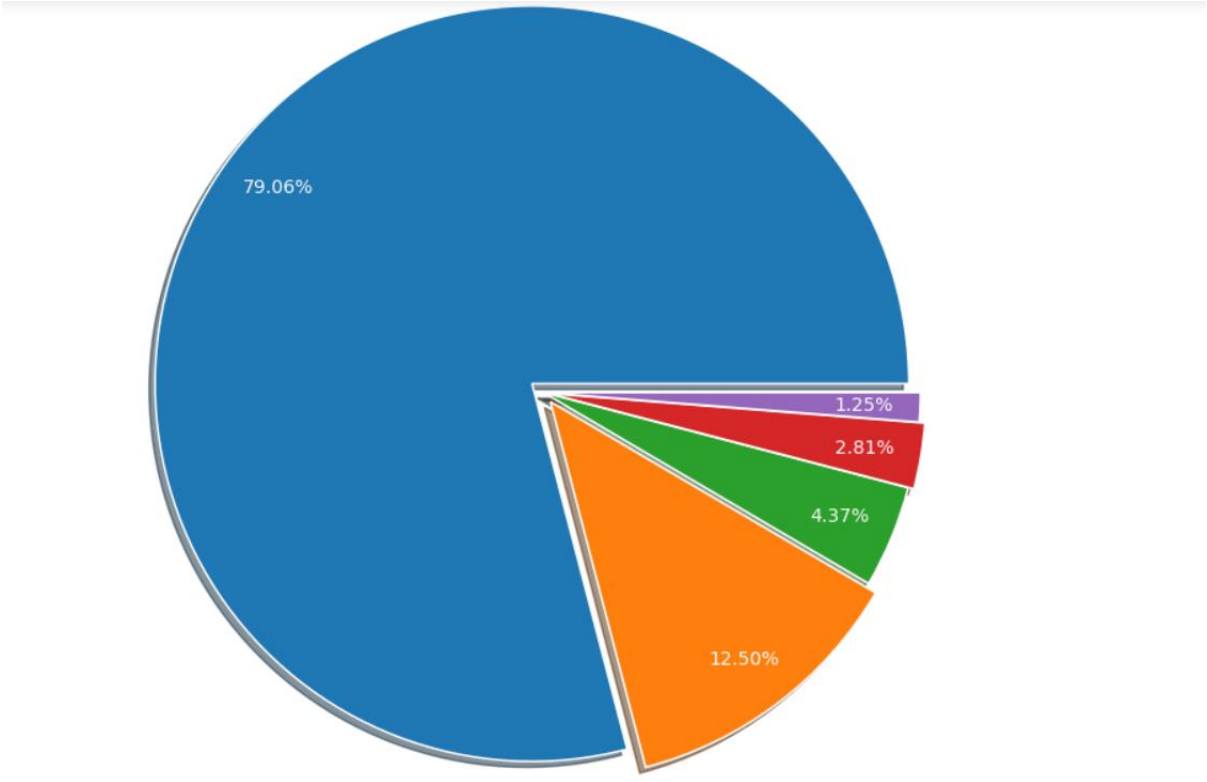


Figure 2:



These two figures are showing the frequencies of the star ratings in figure 1 and in the second figure it is showing the percentage of the reviews based on the ratings per total star reviews.

A summary statistic of the star ratings was carried out as the rest was string data for which the calculating all these was either not possible or was not worth. The results of these ratings are shown below in figure 3:

Figure 3:

mean	3.952223
std	1.233030
min	1.000000
25%	3.000000
50%	4.000000
75%	5.000000
max	5.000000

Review of the NLP techniques and why they have been used:

For any NLP project there are variety of techniques that are available that can help us in getting meaningful insight from the data. In our project we have used Sentiment analysis and Latent Semantic Indexing as our core techniques and the Machine learning techniques that follow them is Naïve Bayes in the case of Sentiment Analysis and Logistic Regression in the case of Semantic Indexing.

Firstly, the justification to use the Sentiment Analysis is carried from the fact that in todays business world people are getting more loaded with information and a peer effect always play a vital role in making buying decisions. Customers are able to communicate their opinions and feelings more openly than ever before, therefore understanding people's emotions is critical for organisations. It is really difficult for a person to look through each line and find the emotion that represents the user experience. With the advancement of technology, we can now analyse consumer feedback automatically, from survey replies to social media chats, allowing firms to listen to their customers and adjust products and services to match their demands. In our case as well if a customer is thinking of buying such an expensive product for his/her operating machine then they will definitely rely on what the already using consumers have to say about this product. So, in order to understand how the brands' product is working in a market companies can easily make decision based on key insights of word text that is not quantifiable in its core essence but when processed with these kinds of techniques offer valuable customer perception of the brand and the product itself. (Milosz Krasiński, 2019)

For the reason the Semantic Indexing is used the basic intention was to see what are the key meaning of the words that are used by the consumers in the review section. Semantic analysis can help machines understand text automatically, which contributes to the larger goal of translating information–that potentially valuable piece of customer feedback or insight in a review or in a customer experience log–into the realm of business intelligence for customer support, corporate intelligence, or knowledge management. Our whole focus is on capturing the topic by just making the machine understand certain words. This theme or issue is sometimes referred to as a latent dimension. Because we can't perceive the dimension explicitly, it's latent. Rather, we only grasp it after reading the text. This signifies that the majority of the words are semantically related to one another in order to express a topic. So, if terms appear with changing frequency in a collection of papers, it should illustrate how various individuals strive to express themselves using different words and distinct subjects or themes. (Subhasis Dasgupta, 2021) In our case a function is created to pull

out 10 similar reviews from the corpus by providing the model with only one review based on similar theme.

With all the justification provided related to why any of the NLP techniques is being used now we move onto the part where we describe the sequence on how we have build the models and what kind of pre-process have been carried out on the input data to make sure that they produce the results that we expect from the model. Firstly, after cleaning of the of the initial web crawled corpus the first thing that was carried out was the stemming and lemmatization of the remaining corpus. Stemming was carried out in order to remove affixes from words in order to obtain their basic form. Whereas, lemmatization was carried out to get the lemma that is the output of lemmatization which offers us the root word rather than the root stem as in the case of stemming. After lemmatization, we will get a valid word that means the same thing. Then frequencies of the words that occur multiple times as well as once in the whole sentences was done followed by creating N-grams, Bi-grams and Tri-grams of bag of words for future processing. This was done as reading the whole sentences in a review that is clean still is a time-consuming task even for the machine so in order to make the assessment easy in the upcoming models that are going to be used these grams were created with their respected visualizations.

Data for the NLP techniques:

With now the project moving onto the final stage of the process we are going to talk about how the initial harvested data is further pre-processed and visualized in order to get a final hold of the data before committing to the original NLP processes. In the initial stages stemming and lemmatization was carried out in order to cut down the affixes and also to get the root word from the whole corpus as shown in the Code Chunk 9 below.

Code Chunk 9:

```
wn_lem = nltk.wordnet.WordNetLemmatizer()
stemmer = nltk.stem.PorterStemmer()
def lemmatization(reviews):
    lemmatized_reviews = []

    for review in reviews:
        # Tokenization
        tokens = word_tokenize(review)

        for index in range(len(tokens)):
            tokens[index] = wn_lem.lemmatize(tokens[index])
            tokens[index] = stemmer.stem(tokens[index])

        lemmatized = ' '.join([token for token in tokens])
        lemmatized_reviews.append(lemmatized)

    return lemmatized_reviews
```

Further, word frequencies count was carried out in the next step in order to clean out words that have occurrence frequency less than 1 and will have limited impact on the future processes by using the code as shown in the Code Chunk 10.

Code Chunk 10:

```
from collections import Counter

frequencies = Counter(' '.join([review for review in clean_reviews]).split())

frequencies.most_common(10)

# Words with least frequency that is 1

singletons = [k for k, v in frequencies.items() if v == 1]

singletons[0:10]
```

Output:

```
['flawless',
 'lid',
 'capton',
 'resist',
 'lead',
 'comparison',
 'swear',
 'decis',
 'liter',
 'fli']
```

```
# This function will remove words with less frequencies

def remove_useless_words(reviews, useless_words):

    filtered_reviews = []

    for single_review in reviews:

        tokens = word_tokenize(single_review)
```

After this cleaning a count vector is calculated to see the words with the most occurrence of frequency and to analyse if they sound important to a common understanding. Code Chunk 11 shows the code and output for this execution.

Code Chunk 11:

```
# count vectoriser tells the frequency of a word.

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df = 1, max_df = 0.9)

X = vectorizer.fit_transform(clean_reviews)

word_freq_df = pd.DataFrame({'term': vectorizer.get_feature_names(),
'occurrences':np.asarray(X.sum(axis=0)).ravel().tolist()})

word_freq_df['frequency'] = word_freq_df['occurrences']/np.sum(word_freq_df['occurrences'])

word_freq_df = word_freq_df.sort_values(by="occurrences", ascending = False)

word_freq_df.head()
```

Output:

	term	occurrences	frequency
537	ssd	240	0.043980
533	speed	155	0.028404
276	instal	111	0.020341
291	laptop	109	0.019974
477	samsung	96	0.017592

These all steps make up for the last data-wrangling that is to be carried out on this harvested corpus before we execute the NLP processes in order to get the insights from the data.

There is a further effort to create some sort of visualizations from the available data so a further clarity can be attained in which things to focus on when carrying out the NLP tasks. A bi-gram and a tri-gram process is carried out on the cleaned data frame and bag of words are created to keep aside visual evidence for assessing the NLP models' results. These bag of words we can identify which are the most important words that are bagged together and then compare it with our results.

Following codes have been executed and their respective outputs are shown as figure 3 and 4 for each bi-grams and tri-grams in code chunk 12.

Code Chunk 12:

```
def get_top_n2_words(corpus, n=None):
  vec1 = CountVectorizer(ngram_range=(2,2),  #for tri-gram, put ngram_range=(3,3)
  top3_words = get_top_n3_words(clean_reviews, n=200)
  top3_df = pd.DataFrame(top3_words)
  top3_df.columns=["Tri-gram", "Freq"]
  #Tri-gram plot
  import seaborn as sns
  top20_trigram = top3_df.iloc[0:20,:]
  fig = plt.figure(figsize = (10, 5))
  return words_freq[:n]
top2_words = get_top_n2_words(clean_reviews, n=200) #top 200
top2_df = pd.DataFrame(top2_words)
top2_df.columns=["Bi-gram", "Freq"]
top2_df.head()
#Bi-gram plot
import matplotlib.pyplot as plt
import seaborn as sns
top20_bigram = top2_df.iloc[0:20,:]
fig = plt.figure(figsize = (10, 5))
plot=sns.barplot(x=top20_bigram["Bi-gram"],y=top20_bigram["Freq"])
plot.set_xticklabels(rotation=45,labels = top20_bigram["Bi-gram"])
#Tri-gram
def get_top_n3_words(corpus, n=None):
  vec1 = CountVectorizer(ngram_range=(3,3),
       max_features=2000).fit(corpus)
  bag_of_words = vec1.transform(corpus)
  sum_words = bag_of_words.sum(axis=0)
  words_freq = [(word, sum_words[0, idx]) for word, idx in
           vec1.vocabulary_.items()]
  words_freq =sorted(words_freq, key = lambda x: x[1],
          reverse=True)
```

Figure 3:

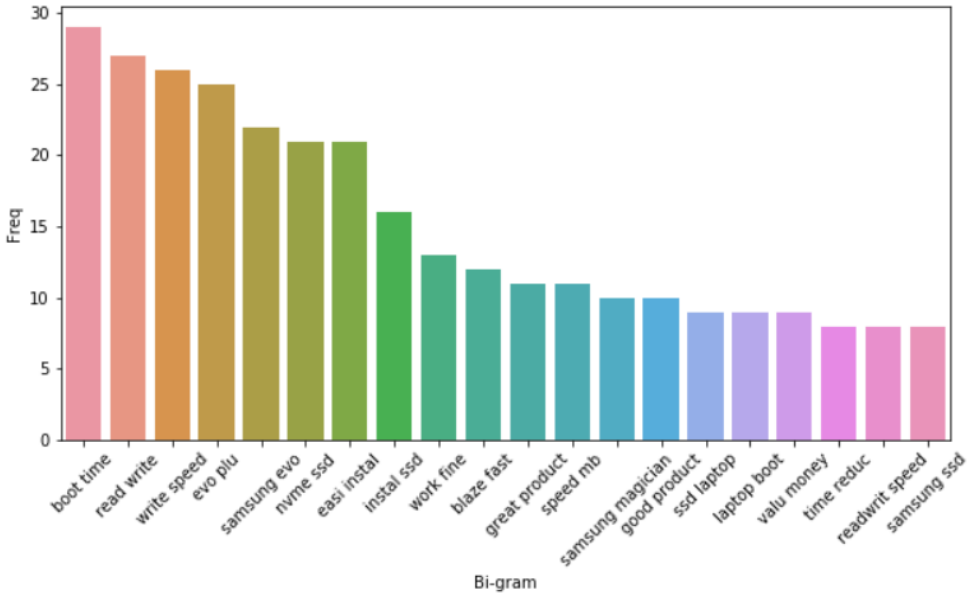
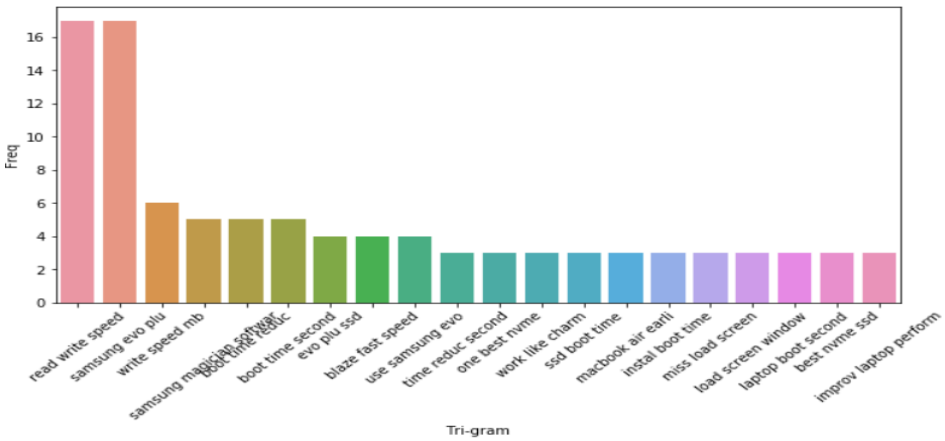


Figure 4:



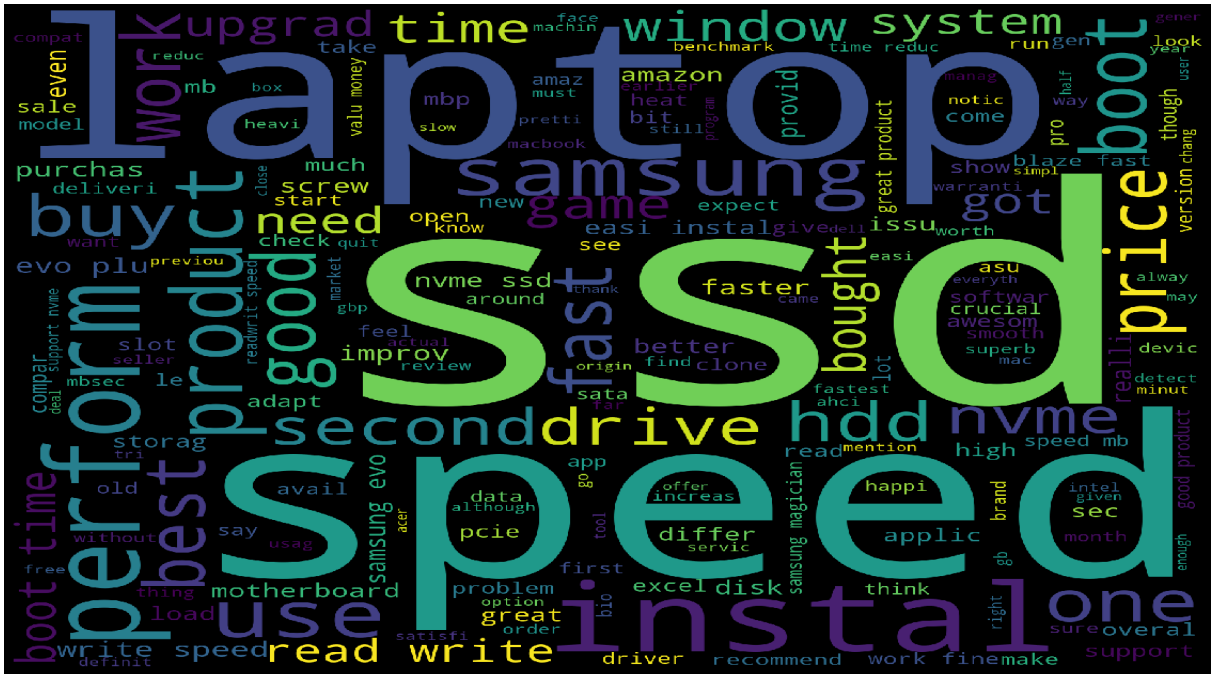
Lastly, a wordcloud of the whole corpus was also created in order to see what are the major words in the remaing corpus that have significance due to their occurance frequecny or importnace as an individual variable.

Code Chunk 13 shows the execution code and figure 5 shows the output result.

Code Chunk 13:

```
string_Total = " ".join(clean_reviews)

#wordcloud for entire corpus

plt.figure(figsize=(20, 20))

from wordcloud import WordCloud

wordcloud_stw = WordCloud(

        background_color= 'black',

        width = 1800,

        height = 1500

        ).generate(string_Total)

plt.imshow(wordcloud_stw)

plt.axis("off")

plt.show()
```

Figure 5:



With all this done now we have the bagged words and all the frequency important words to carry out our NLP modelling and develop our Machine learning processes.

NLP tasks Implementation:

Now we move onto the NLP tasks that are implemented in order to achieve the final outcomes of our whole assessment. First we carry out the sentiment analysis of our cleaned data frame where we first calculate the subjectivity and polarity of the data frame using the textblob package in python to see how subjective the reviews are and whether they are positive or negative in context based on their polarity.

The next Code Chunk 14 shows the code that is used to execute the process and the output subsequently shows the results achieved executing those commands.

Code Chunk 14:

```
from textblob import TextBlob

# Get Subjectivity of each tweet

def getSubjectivity(tweet):

    return TextBlob(tweet).sentiment.subjectivity


# Get polarity of each tweet

def getPolarity(tweet):

    return TextBlob(tweet).sentiment.polarity

sentiment_df = pd.DataFrame(clean_reviews, columns=["reviews"])

sentiment_df["Subjectivity"] = sentiment_df["reviews"].apply(getSubjectivity)

sentiment_df["Polarity"] = sentiment_df["reviews"].apply(getPolarity)

sentiment_df.head()
```

Output:

	reviews	Subjectivity	Polarity
0	brand brand alway read write speed ssd success...	0.588148	0.276667
1	lot research ssd decid give tri right make hug...	0.511905	0.328571
2	acer nitro detect first boot instal secondari ...	0.200000	0.070833
3	macbook dell hdd god speed slow new dell lapto...	0.425758	-0.010606
4	guy suggest buy product samsung product first ...	0.673611	-0.187500

Further, we model the sentiments and see if the model is performing according to what it needs to do and we can see in Figure 6 that the model correctly identifies the positive and negative reviews based on the polarity and we can describe the subjectivity of the review easily based on its subjectivity coefficient value. This is all possible based on the TFIDF technique we used earlier that has signified the importance of the word in the document and corpus.

Code Chunk to execute it is 15 and is as follows:

Code Chunk 15:

```
# Funciton to compute Sentiment Analysis
def getAnalysis(score):
    if score < 0:
        return "Negative"

    elif score == 0:
        return "Neutral"

    else:
        return "Positive"
sentiment_df["Analysis"] =
sentiment_df["Polarity"].apply(getAnalysis)
sentiment_df.head()
```

Figure 6:

	reviews	Subjectivity	Polarity	Analysis
0	brand brand alway read write speed ssd success...	0.588148	0.276667	Positive
1	lot research ssd decid give tri right make hug...	0.511905	0.328571	Positive
2	acer nitro detect first boot instal secondari ...	0.200000	0.070833	Positive
3	macbook dell hdd god speed slow new dell lapto...	0.425758	-0.010606	Negative
4	guy suggest buy product samsung product first ...	0.673611	-0.187500	Negative

Now after this execution we execute the Naïve Bayes ML technique for that we have associated with the Sentiment Analysis where the star ratings are classified based on their meaning that were made based on TFIDF vectors that gave meaning to each word and the output from the subjectivity and polarity analysis that showed the values associated with each review. In its nature it's a multinomial Naïve Bayes as shown in the code output as well and prediction accuracy is there after carried out based on the built model to see the accuracy and other measures of the validation.

On the test and train split the percentage used was based on industry best practice of 70% and 30% for train and test respectively.

The Code Chunk 16 below shows the commands used to execute the ML Naïve Bayes.

Code Chunk 16:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(reviews['review_text'], reviews['rating'], \
                                   test_size=0.3, random_state=0)


print('Load %d training examples and %d validation examples. \n'
%(X_train.shape[0],X_test.shape[0]))
print('Show a review in the training set : \n', X_train.iloc[10])
X_train,y_train
def cleanText(raw_text, remove_stopwords=False, stemming=False, split_text=False, \
        ):
  '''
  Convert a raw review to a cleaned review
  '''
  text = BeautifulSoup(raw_text, 'html.parser').get_text()
  letters_only = re.sub("[^a-zA-Z]", " ", text)
  words = letters_only.lower().split()

  if remove_stopwords:
    stops = set(stopwords.words("english"))
    words = [w for w in words if not w in stops]

  if stemming==True:

    stemmer = SnowballStemmer('english')
    words = [stemmer.stem(w) for w in words]

  if split_text==True:
    return (words)

  return( " ".join(words))
```

```
X_train_cleaned = []
X_test_cleaned = []


for d in X_train:
   X_train_cleaned.append(cleanText(d))
print('Show a cleaned review in the training set : \n',  X_train_cleaned[10])


for d in X_test:
   X_test_cleaned.append(cleanText(d))
countVect = CountVectorizer()
X_train_countVect = countVect.fit_transform(X_train_cleaned)
mnb = MultinomialNB()
mnb.fit(X_train_countVect, y_train)
def modelEvaluation(predictions):
   print ("\nAccuracy {:.4f}".format(accuracy_score(y_test, predictions)))
   print("\nClassification report : \n", metrics.classification_report(y_test, predictions))
predictions = mnb.predict(countVect.transform(X_test_cleaned))
modelEvaluation(predictions)
```

For the second task the Latent Semantic Analysis a new TFIDF model was built before execution of the LSI and then from the genism package Lsi Model was imported that was then fitted to the recently built TFIDF model beforehand. There was also a dictionary of words that was created that was later tokenized for usage in the TFIDF model. Furthermore, there was a function created that brought us the top 10 similar reviews based on a similar topic or theme that based on LSI model itself.

The code to execute the intended model can be found in the following Code Chunk 17.

Code Chunk 17:

```
token_reviews = []
for review in clean_reviews:
    token_reviews.append(word_tokenize(review))


dictionary = corpora.Dictionary(token_reviews)
dictionary.items()
dictionary = corpora.Dictionary(token_reviews)
 for key in dictionary:
        print(key, dictionary[key])
corpus = [dictionary.doc2bow(review) for review in token_reviews]
corpus
##TFIDF model
tfidf_model = models.TfidfModel(corpus)
corpus_tfidf = tfidf_model[corpus]
corpus_tfidf
##LSI Modelling
from gensim.models.lsimodel import LsiModel
from gensim import similarities
lsi_model = LsiModel(corpus = corpus_tfidf, id2word = dictionary, num_topics = 400)
index = similarities.MatrixSimilarity(lsi_model[corpus])
def text_lsi(new_text, num = 10):
    text_tokens = word_tokenize(new_text)
    new_vec = dictionary.doc2bow(text_tokens)
    vec_lsi = lsi_model[new_vec]
    similars = index[vec_lsi]
    similars = sorted(enumerate(similars), key = lambda item: -item[1])


    return [(s, clean_reviews[s[0]]) for s in similars[:num]]
text_lsi(clean_reviews[100])
```

Following all the outputs that were received from the code chunk 17 the later step of inducing an ML technique came in where logistic regression was carried out based on the themes that were identified from the LSI. For the Logistic regression a new TFIDF model was created based on the themes identified with coefficients of most and minimum in nature were calculated. Lastly, prediction accuracy of the model was tested based on the test data we had to see the accuracy of the model in predicting the star rating of the reviews.

The Code Chunk 18 shows the whole command input to develop the ML technique and the prediction assessment of the model.

Code Chunk 18:

```
tfidf = TfidfVectorizer(min_df=5)

X_train_tfidf = tfidf.fit_transform(X_train)



# Logistic Regression

lr = LogisticRegression()

lr.fit(X_train_tfidf, y_train)

feature_names = np.array(tfidf.get_feature_names())

sorted_coef_index = lr.coef_[0].argsort()

print('\nTop 10 features with smallest coefficients :\n{}\n'.format(feature_names[sorted_coef_index[:10]]))

print('Top 10 features with largest coefficients : \n{}'.format(feature_names[sorted_coef_index[:-11:-1]]))
```

Output:

```
Top 10 features with smallest coefficients :
['product' 'not' 'no' 'see' 'installing' 'available' 'box' 'magician'
 'getting' 'first']
Top 10 features with largest coefficients :
['good' 'speed' 'performance' 'you' 'for' 'price' 'fast' 'as' 'best' 'my']
```

```
predictions = lr.predict(tfidf.transform(X_test_cleaned))

modelEvaluation(predictions)
```

Lastly, I will discuss about the Hyperparameters that were optimized in Semantic analysis and the codes that were used to execute to increase the performance of the model of the logistic regression by tweaking certain parameters in the model are in Code Chunk 19:

Code Chunk 19:

```
param_grid = {'C': np.logspace(-4, 4, 50),
        'penalty':['l1', 'l2']}
clf = GridSearchCV(LogisticRegression(random_state=0), param_grid,cv=5, verbose=0,n_jobs=-1)
best_model = clf.fit(X_train,y_train)
print(best_model.best_estimator_)
print("The mean accuracy of the model is:",best_model.score(X_test,y_test))
```

By tuning the base parameters, the model accuracy increased to 94% from the initial 75%.

Outputs and Results:

In this last section I will discuss about the outputs and the interpretation that is derived from those outputs. All the codes executed without any hinderance and each and every model produced output in relation to what was expected from it.

Firstly, we will discuss about the outputs from the NLP tasks that were carried out. In the sentiment analysis we got the values of polarity and subjectivity and when based on those the call for the positive reviews and the negative reviews was called out the resultant showed quiet appropriate results as expected from the model. In the Output below it can be seen that the positive comments that were produced are accurate in description.

Output:

```
All Positive Reviews are:

1 )  great way improv overal perform highli recommend still use old slow hard drive
slow perform instal ssd lenovo ideapad ikb realli happi

2 )  instal ssd press power boot got overheat boot window via hdd temp ssd go go
show high temp abl boot laptop due issu asu vivobook gen laptop

3 )  compat seri laptop dedic ssd boot time second shut time second previous averag
boot time sec averag shutdown time second instal ssd applic open applic multitask
smooth without

4 )  fast better also buy version time new laptop bad thing give big issu

5 )  blaze fast perform price bit high compar other perform easi instal boot time
second window instal time minut offic instal minut video transfer within ssd differ
second

6 )  boot sec way fast readwrit valu
```

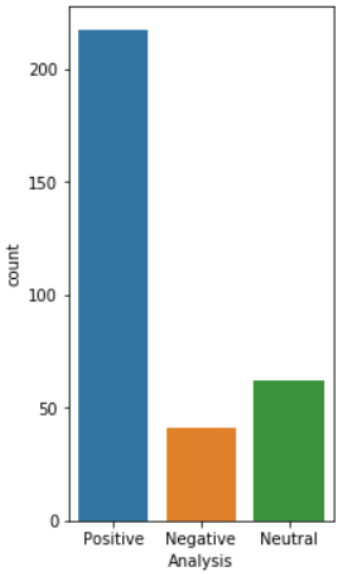
For the negative reviews based on the Sentiment analysis we can analyse that the model brought up accurate results as well as shown in the output below:

Output:

```
All Negative Reviews are:

1 )  product ever brought samsung alreadi corrupt product seller know sold
bad product veri upset also replac product

2 )  fake product deliveri ssd includ

3 )  match product defect fake

4 )  purchas product upgrad one issu packag includ tighten screw use sad
manag get pair tighten screw

5 )  fake product reciev serial number mention box match serial number show
samsung magician softwar also magician softwar detect ssd realli disappoint

6 )  difficult find tighten ssd one came pleas ensur one order drive
```

From the sentiment analysis a visualisation was also made showing the frequency of the positive, neutral and negative reviews which as shown in figure 6 validates the initial EDA outputs that we generated in the initial second task. The pie chart formed showed us that the 5 star ratings were in higher percentage compared to lesser star ratings which validates our output from the figure 6.

Figure 6:



In the latent Semantic Indexing the function that was created to produce 10 similar themed reviews also brought us the accurate results. The output below shows the results from calling that function.

Output:

Based on the following review in consideration:

```
'laptop slow without ssd even ram boot took almost min instal bare take sec
boot laptop softwar open second great ssdthe con bit expens'
```

Results of top 10 similar reviews

```
[((100, 1.0),
  'laptop slow without ssd even ram boot took almost min instal bare take
sec boot laptop softwar open second great ssdthe con bit expens'),
 ((64, 0.4487425),
  'bought game laptop processor slow boot almost min took internet came
know ssd improv laptop speed bought samsung evo insert pci slot clone hdd
ssdnow laptop boot start use applic sec buy laptop support nvme type ssd
use youtub googl get know laptop buy valu money'),
 ((310, 0.39937794), 'quick boot time second easili load laptop softwar'),
 ((158, 0.39849284),
  'ssd amaz easi instal instal catch laptop come screw put ssd place caus
worri laptop boot second everyth super fast'),
 ((147, 0.38452327),
  'compat seri laptop dedic ssd boot time second shut time second previous
averag boot time sec averag shutdown time second instal ssd applic open
applic multitask smooth without'),
 ((23, 0.38150966),
  'amaz ssd laptop hdd boot time way slower laptop slower went ssd instal
ssd laptop boot le sec app program open faster file copi speed also improv
overal ssd improv perform laptop laptop pcie gen port would get speed
around mbsec still way faster pcie gen would get mbsec depend spec'),
```

This output only shows a part of the whole result.

Moreover, now we discuss the quality metrics of the NLP/ML tasks that were carried out. The resultant output showed us that for the Naïve Bayes had an Accuracy of 71% where as the logistic regression in its base form had an accuracy of 75%. With hyperparameter tuning the accuracy of the logistic model was increased to 94% that in its entirety is a very well balanced score keeping in mind that the whole project was based on real time raw data that had no alterations or biases involved in it.

The output below shows the results from Naïve Bayes, Regression and tuned Regression:

Naïve Bayes:

Accuracy 0.7188

Classification report :

	precision	recall	f1-score	support
2.0	0.00	0.00	0.00	1
3.0	0.00	0.00	0.00	1
4.0	0.00	0.00	0.00	6
5.0	0.74	0.96	0.84	24
accuracy			0.72	32
macro avg	0.19	0.24	0.21	32
weighted avg	0.56	0.72	0.63	32

Logistic Regression:

```
Accuracy 0.7500

Classification report :
               precision    recall  f1-score   support

         2.0       0.00      0.00      0.00         1
         3.0       0.00      0.00      0.00         1
         4.0       0.00      0.00      0.00         6
         5.0       0.75      1.00      0.86        24

    accuracy                           0.75        32
   macro avg       0.19      0.25      0.21        32
weighted avg       0.56      0.75      0.64        32
```

Tuned Regression:

```
LogisticRegression(C=10000.0, random_state=0)
The mean accuracy of the model is: 0.9420718191222107
```

In all the metrics its quite evident that the model was very successful in classifying the best 5-star reviews be it Naïve Bayes or Logistic Regression.

Conclusion:

In conclusion I would sum up that all the models that were created and output achieved it is safe to say that if Samsung is looking to see what its customers are going through after using their products, then the company can easily see that their product is doing quiet well in the market as the customers have a healthy experience with the product they are buying. At the same time their customer services team can also build a lot on how to make sure the customers that are not satisfied with the product can be better assisted and serviced in order to bring them also in the club of happy customers.

This whole project gives key insight on what are the factors that are making Samsung's' product great by identifying the words that people use to express their experience with the product hand in hand with insight about what particular problems the consumers are facing while using the product using this word/phrase identifier.

References:

- *India Consumer Electronics Market Size & Share | Industry Report, 2025*. (n.d.). Www.grandviewresearch.com. https://www.grandviewresearch.com/industry-analysis/india-consumer-electronics-market
- Krasiński, M. (2019, June 26). *What is Sentiment Analysis, and Why is it Important?* Mondovo Blog. https://www.mondovo.com/blog/what-is-sentiment-analysis-and-why-is-it-important/
- Dasgupta, S. (2021, September 16). *Latent Semantic Analysis and its Uses in Natural Language Processing*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/09/latent-semantic-analysis-and-its-uses-in-natural-language-processing/