CHRIST
(DEEMED TO BE UNIVERSITY)
PUNE LAVASA CAMPUS
The Hub of Analytics

Query-based comparative Analysis of a Particular Product on Amazon and Flipkart & Sentiment Analysis on Desired Product

Research Project Report

Shreyansh Padarha
(21112026)
3 BSC DS A

Under the Guidance of
Prof Naived George Eapen
Department of Data Science

A Project Report submitted in Partial Fulfilment of the requirements for the award of Degree of Bachelor of Science (Data Science) of
CHRIST (Deemed to be University), Pune Lavasa Campus - 'The Hub of Analytics'

2022 - 23

CHRIST
(DEEMED TO BE UNIVERSITY)
PUNE LAVASA CAMPUS
The Hub of Analytics

CERTIFICATE

This is to certify that the research project titled Query-based Comparative Analysis of a Particular Product on Amazon and Flipkart & Sentiment Analysis on Desired Product is a Bonafede record of work done by Shreyansh Padarha (21112026) of 3 BSC DS, CHRIST (Deemed to be University), Pune Lavasa Campus, in partial fulfilment of the requirements of III Semester BSc (Data Science) during the year 2022-2023.

Dr Lija Jacob
Head of the Department

Naived George Eapen
Project Guide

Table of Contents

[1] Introduction

Customer confusion has been on the uprising for the past decade, this psychological construct can be credited to the rampant e-commerce industry. In 2021, e-retail sales crossed a mammoth 5.2 trillion USD worldwide. Although the numbers can be assumed to be bloated, as a customer, the plethora of e-commerce websites to choose from has made purchasing decisions harder. Ill-informed decisions regarding buying a product can lead to mistrust in online shopping or financial disasters. It's imperative to keep the customers well-informed to keep them satisfied, it also saves ample of time while purchasing a product.

[2] Background

The three underlying principles of e-confusion or customer confusion are similarity confusion, overload confusion and unclarity confusion. But, in most developing countries like India, the biggest cause of consumer swingers' is product price, rating, reviews and of course availability. Keeping this in mind, the project is aimed at building a thorough application to help users in deciding which product to buy from, and more importantly where from. The added feature of generating a sentiment analysis of the reviews made by customers, provides an in-depth understanding of the position of customers regarding the product. Opinion mining was an agenda that was exploited during the project, it helped in determining whether the overall customer experience of the customer is positive, neutral or negative in nature.

[3] Problem Statement

This project attempts to combine data analytics methodologies like KDD (Knowledge Discovery Database) and comparative analysis techniques to tackle an ever-lasting conundrum, that is settling the question, which platform is the best to buy your desired product from? Considering time constraints, the comparisons were limited to Amazon and Walmart subsidiary Flipkart. The same is done by web scraping the requested products query from Amazon and Flipkart, and further web-scrapping product reviews, enabling individual product-based sentimental analysis (Amazon Only).

[4] Data Sources and Collection

The impromptu response nature of the project required an on-demand and customisable dataset generation, every time the user seeks a product. To tackle the complexity, we created three modules of web-scrapping for specific purposes and generating specific datasets.

The sources of the web-scrapped data were Amazon and Flipkart's website. The collection involved a generalised algorithm of web scrapping products and their information.

The reviews scrapped module, went through all the comments of the product and, post data scrubbing, storing the information in a dataset. Post scrapping the reviews the sentiment analysis module created by me, added 8 columns based on a ROBERTA and VADER model

sentiment scores. These scores helped in generating an aggregate opinion (polarity) representative of all the customers.

There were two main datasets that were created through the process.

- Web Scrapping Amazon (based on query (product given by client)
- Web Scrapping Flipkart (based on query (product given by client)
    - The above datasets, are merged and the final product dataset created with the web scrapped data had the variables:

        1. querry_searched
        2. ecommerce_website
        3. product_name
        4. product_price
        5. product_rating
        6. rating_count
        7. poduct_image_url
        8. product_url

- Web Scrapping Comments (based on the product selected from the above dataset) (Amazon Only) .
    - Post applying NLP models like ROBERTA and VADER on the comments scrapped, to generate sentiment scores. The comments scrapped dataset had the variables:

        1. customer_name
        2. review_date
        3. review_title
        4. review_ratings
        5. review_content
        6. review_size
        7. sentiment
        8. roberta_sentiment
        9. vader_sentiment
        10. roberta_neg
        11. roberta_neu
        12. roberta_pos
        13. vader_neg
        14. vader_pos
        15. vader_comp

[5] Tools and Libraries

IDLE Used: Python Shell, PyCharm, Jupyter Notebook

Python modules and libraries used:

- BeautifulSoup
- requests
- pandas
- numpy
- scipy
- seaborn
- random
- matplotlib
- warnings
- time
- datetime
- pltoly
- regex
- re
- smtplib
- Ipython
- nltk.sentiment
- tqdm.notebook
- transformers
- scipy.special (softmax)
- tabulate
- urlib.request
- PIL
- Wordcount

[6] Methodology

The process and methodology involved during the project were a robust and in-depth exploration of various components and fragments of KDD and data analytics.

- Understanding requirements to handle the Problem Statement.
  This first vital step made me realise the complexity of the project. Web-scrapping using Python Libraries had to be learnt from scratch using tutorials and library documentation.

- Designing the architecture of the program
  From the onset, it was understood that the project required an elaborate backend and a sophisticated design to help integrate it with the finesse of the front end. This prompted me to create 3 separate modules and import it into the final file as .py finals. Using concepts from OOPs would result in a simplified and easy to traceback software.

The backend modules include :

- **webscraping.py** module:

  - Web Scrapping Amazon Class (based on query (product given by the client)
  - Web Scrapping Flipkart Class (based on query (product given by the client)

    - Dataset Created with the web scraped data had the variables:

    1. querry_searched
    2. ecommerce_website
    3. product_name
    4. product_price
    5. product_rating
    6. rating_count
    7. poduct_image_url
    8. product_url

  - Web Scrapping Comments Class (based on the product selected from the above dataset) (Amazon Only)

    - Dataset Created with the web scraped data had the variables:

    1. customer_name
    2. review_date
    3. review_title
    4. review_ratings
    5. review_content

- **ProjectVisualisations.py** module:

  - Consists of various functions that help in plotting the amazon vs Flipkart data.
  - Using Plotly, seaborn, and matplotlib to come up with 17 plots for a website vs website analysis.

- **ProductReviewAnalysis.py** module:

  - This file contained was the main file for performing a sentiment analysis upon reviews/comments on a product (currently limited to amazon)

    + Using ROBERTA Model (NLP)
    + Using VADER Model (NLP)
    + A weighted 60:40 approach (Roberta : Vader) to obtain a final sentiment value.
    + 18 Visually Appealing Plots

- Inspecting elements of static HTML pages of Amazon and Flipkart and Scrapping based on trial and error
  The web scraping was done in tandem with the requests module and BeautifulSoup module, this involved requesting the search page based on the user's query. The page retrieved then had to be scouted for required information. This was done with basic knowledge of the inbuilt methods of the libraries being used and on a trial-and-error basis. It was a tedious process as it required checking where the elements were present on the actual page and comparing it in my script.

- Cleaning data and transforming data while it's being scrapped
  A major difference between traditional EDAs and models created by me in the past and this project was, I had to clean and make the data apt while it was being scrapped and before inserting it into the dataset. This involved using regex and string functions and logic to make the data presentable and comprehensible.

- Visualising Amazon vs Flipkart product range for the product
  Using Plotly, seaborn, and matplotlib to paint an effective picture of how the products vary on Amazon and Flipkart for a user's query. I created various plots using user-defined functions and recalled all of them under one visualising parent function.

- Scrapping All Comments based on the item selected
  During, the initial bout of scraping all products from Amazon and Flipkart, the onclick() URL was also stored, in case of further analysis. Based on the product selected for sentiment analysis the URL is used for sending a request, and at the bottom of the product details page on Amazon, we have a "see all reviews" button. A request is sent to that button's onclick() URL, and a loop is used to change the URL page no. and all the reviews are scrapped along with the writer's name, date and review stars given. All of this is stored orderly in a panda's data frame.

- Sentiment Analysis using Vader and Roberta Model
  The comments were analysed with the help of NLTK module and hugging face models. I used two different models (VADER and Roberta) and a weighted 60:40 approach to extract an accurate sentiment for the product.

  VADER (Valence Aware Dictionary for Sentiment Reasoning) is a Model that is sensitive to polarity (positive/negative) and intensity of a emotion. For example words like 'love', 'enjoy', 'happy', 'like' all convey a positive sentiment. Also, VADER is intelligent enough to understand the basic context of these words, such as "did not love" as a negative statement. It also understands the emphasis of capitalization and punctuation, such as "ENJOY*".* The drawback of VADER is that it's optimised for social media, and it doesn't take context or sarcasm into consideration.

ROBERTA is a model that is an enhanced version of Google's BERT model. BERT makes use of a Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms an encoder that reads the text input and a decoder that produces a prediction for the task. ROBERTA has minor tweaks added to BERT. The model was trained with a hugging face 58 million tweets dataset, finetuned for natural language processing.

- Visualising Sentiment Analysis
  The following maps, graphs and comparisons are used to display the results of the sentiment analysis:
  + Word Frequency Chart
  + Sentiment (Positive, Negative, Neutral) Count Plot, Pie Chart
  + Weighted Sentiment Spread (distplot)
  + Roberta vs Vader Model Sentiment Score comparison
  + Review count for the product (Bar graph, Pie Chart)
  + Length of the review vs Product Rating
  + In depth comparison of Vader and Roberta Scores

- Interesting Fact (Learning)
  The project made use of 81 Methods (User-Defined Functions) in total, for efficient and sophisticated coding. The functions and blocks of code were easier to debug as well.

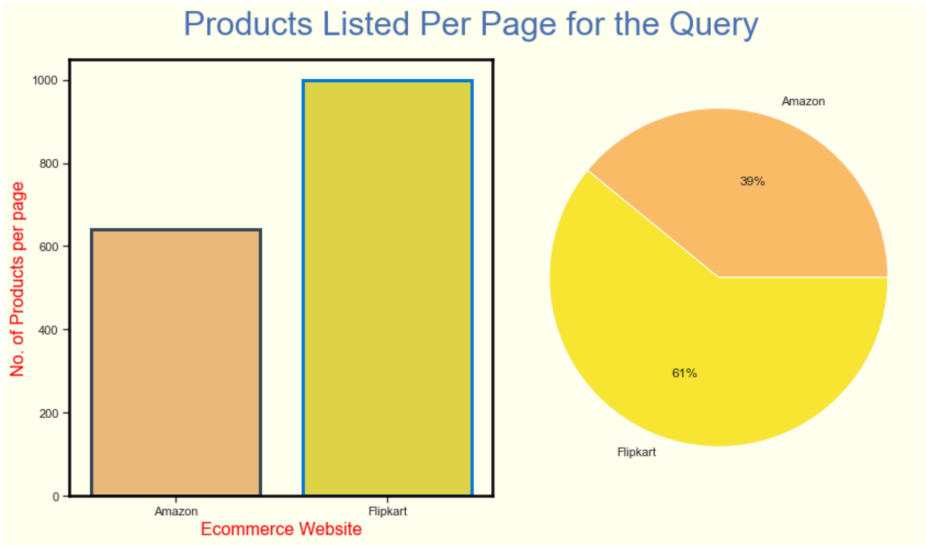
[7] Observations and Findings

The observations are on a per-case basis, as can be seen in the below Classmate Notebook Comparison the results are self-explanatory.

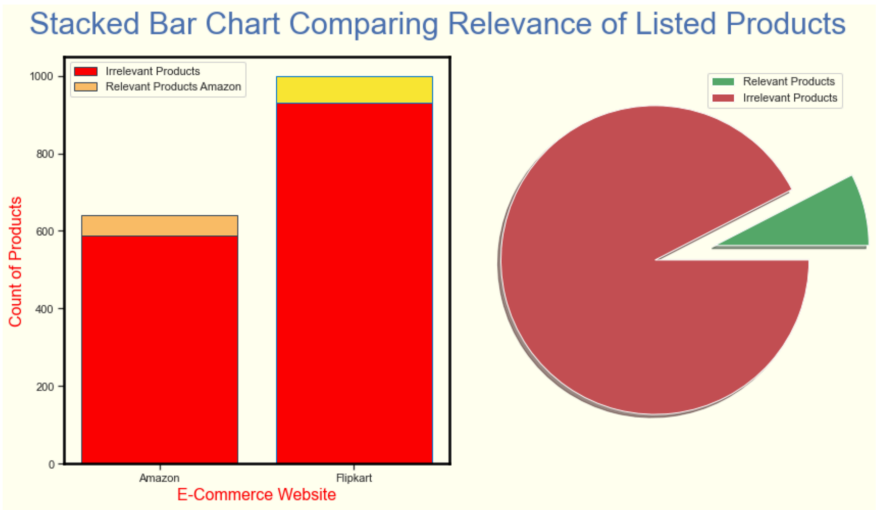
----------**AMAZON vs FLIPKART Comparision**----------

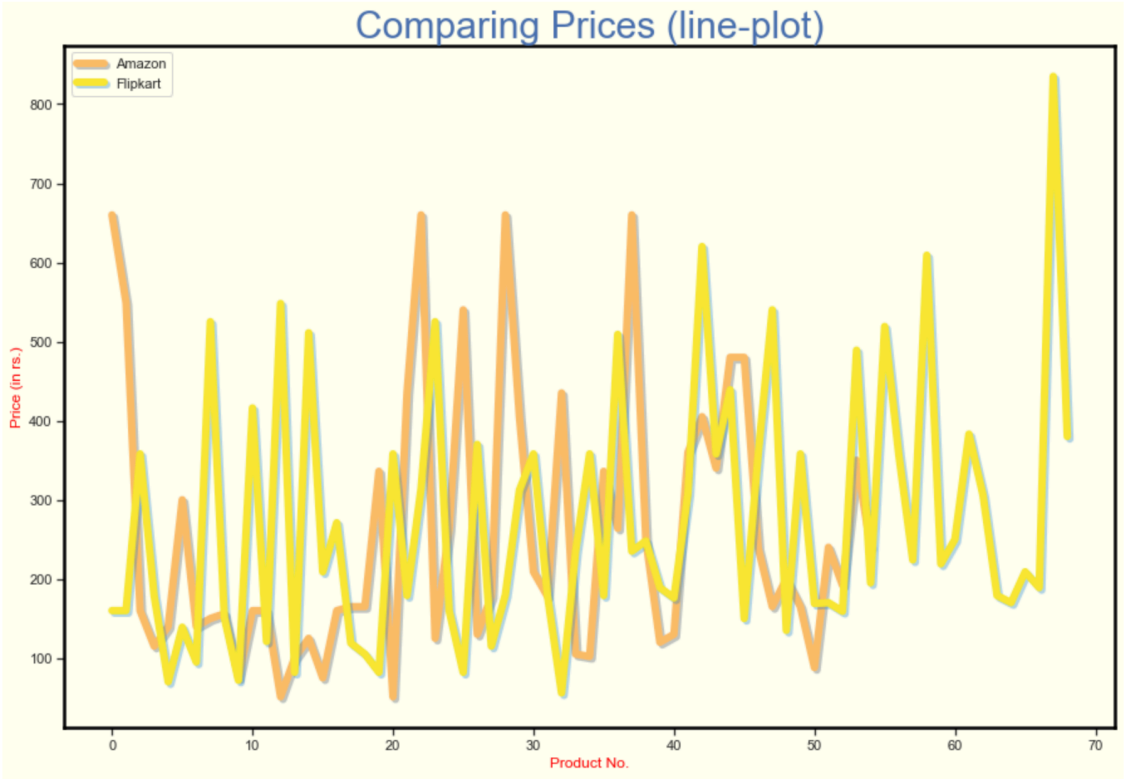
PRODUCT QUERIED : Notebook
BRAND : Classmate

-> Amazon has 642 products listed on each page.
-> Flipkart has 1000 products listed on each page.

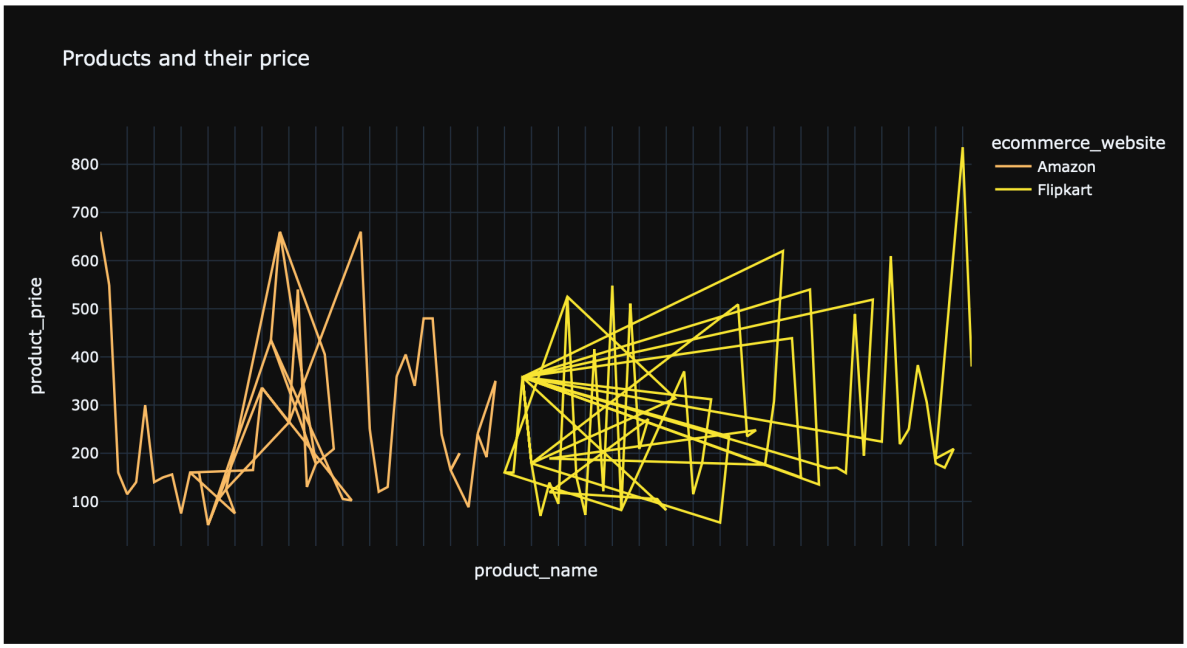


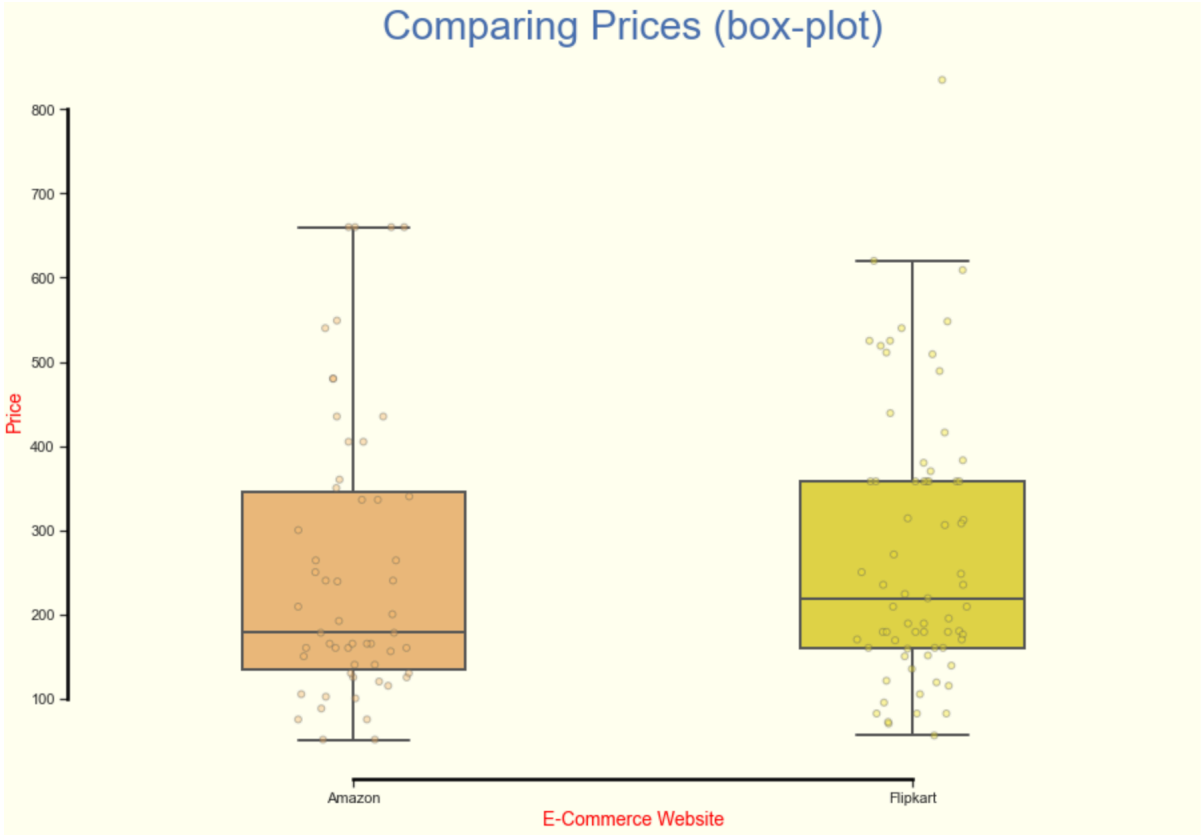
-> Amazon has 642 products listed on each page for the queried product,out of which 587 are of other brands
-> Flipkart has 1000 products listed on each page for the queried product,out of which 69 are of other brands

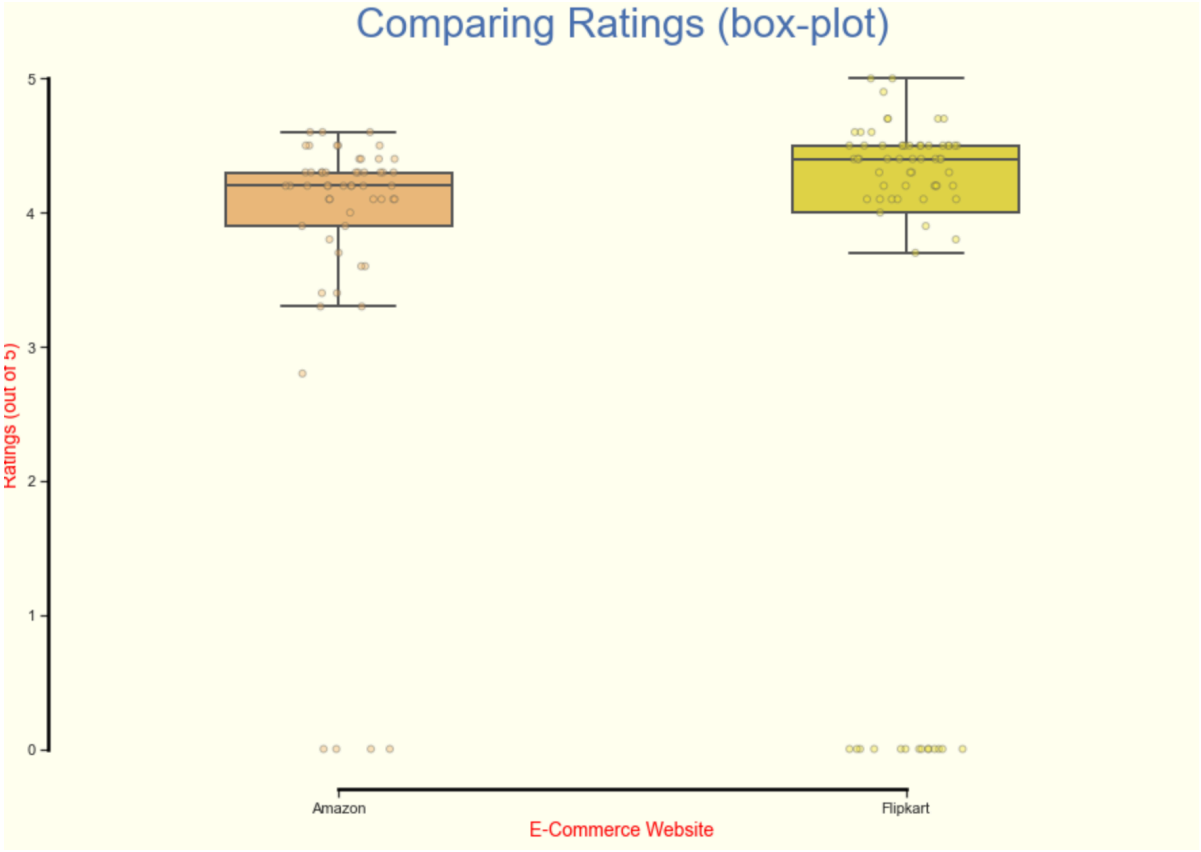





```
Average Product Price for queried product at amazon is :  256.7818181818182
Average Product Price for queried product at flipkart is :  274.8985507246377
```




Comparing Prices (box-plot)
Price
800
700
600
500
400
300
200
100
Amazon
Flipkart
E-Commerce Website

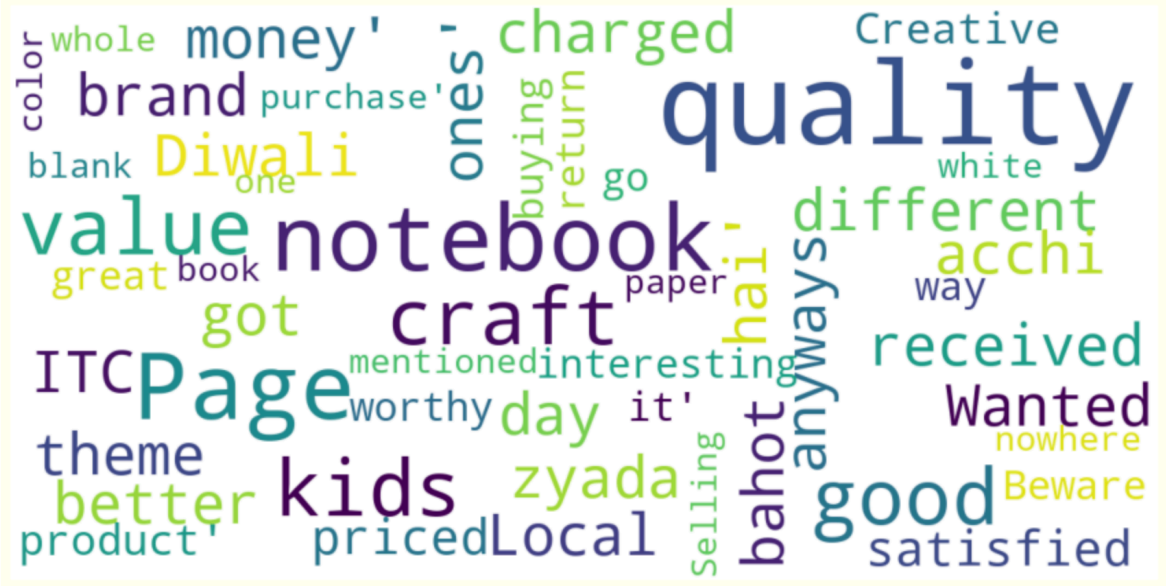


Sentiment Analysis

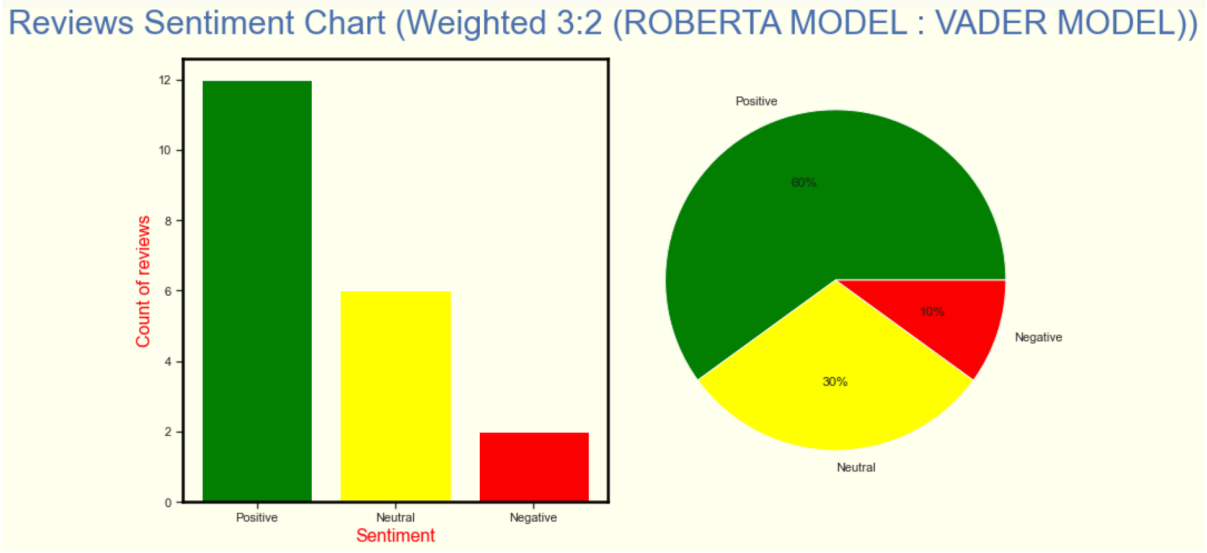
----------PRODUCT DETAILS----------

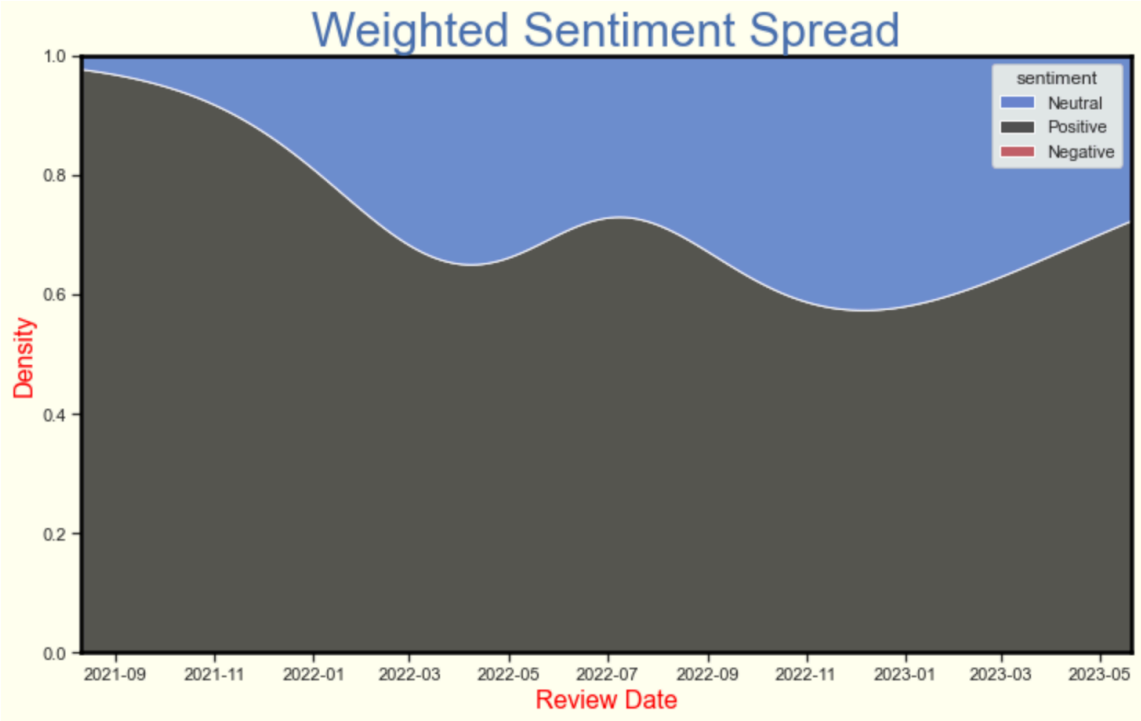
Product Name : Classmate Origami Notebooks - Single Line, 168 Pages, 240 mm x 180 mm - Pack Of 12
Product Price : 660.0
Average Product Rating : 4.3
Total Number of Ratings : 0.0

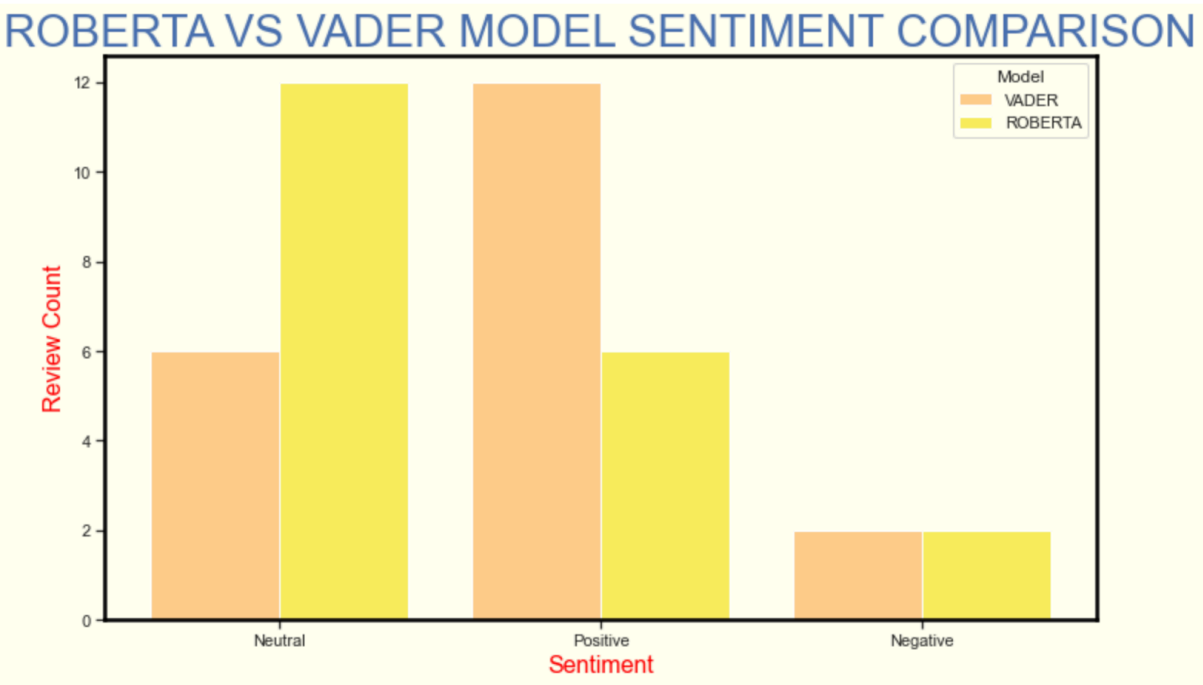
WORD FREQUENCY CHART

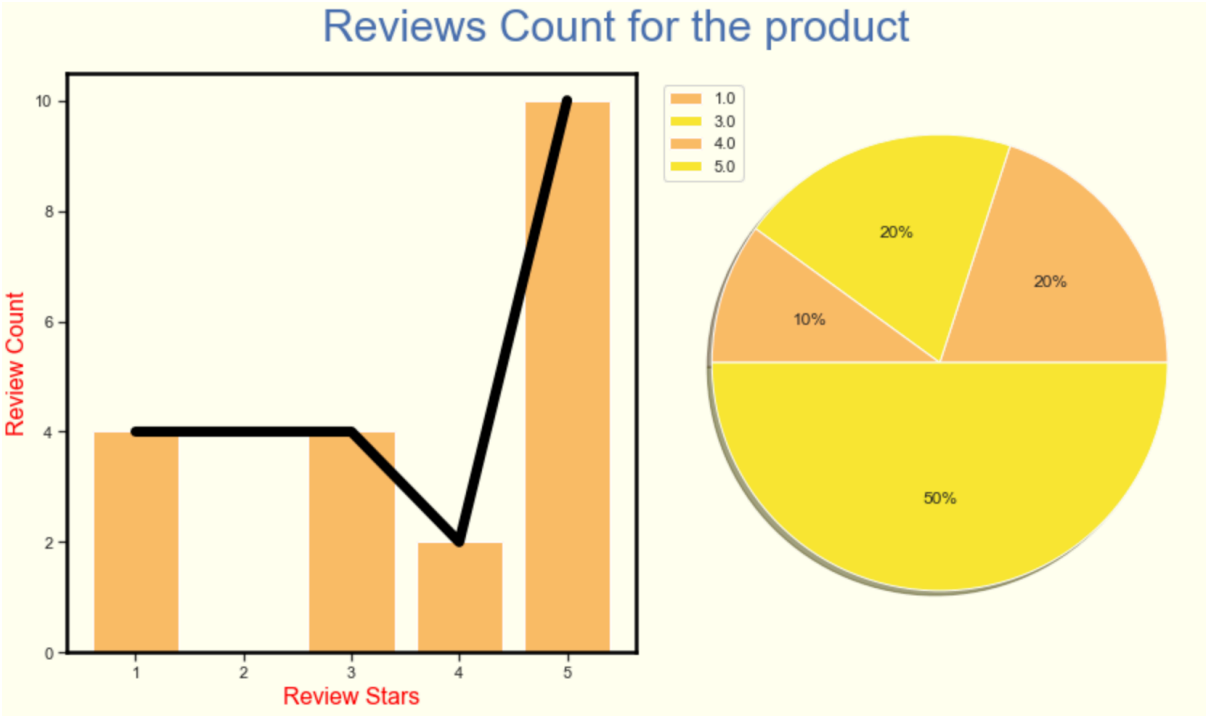


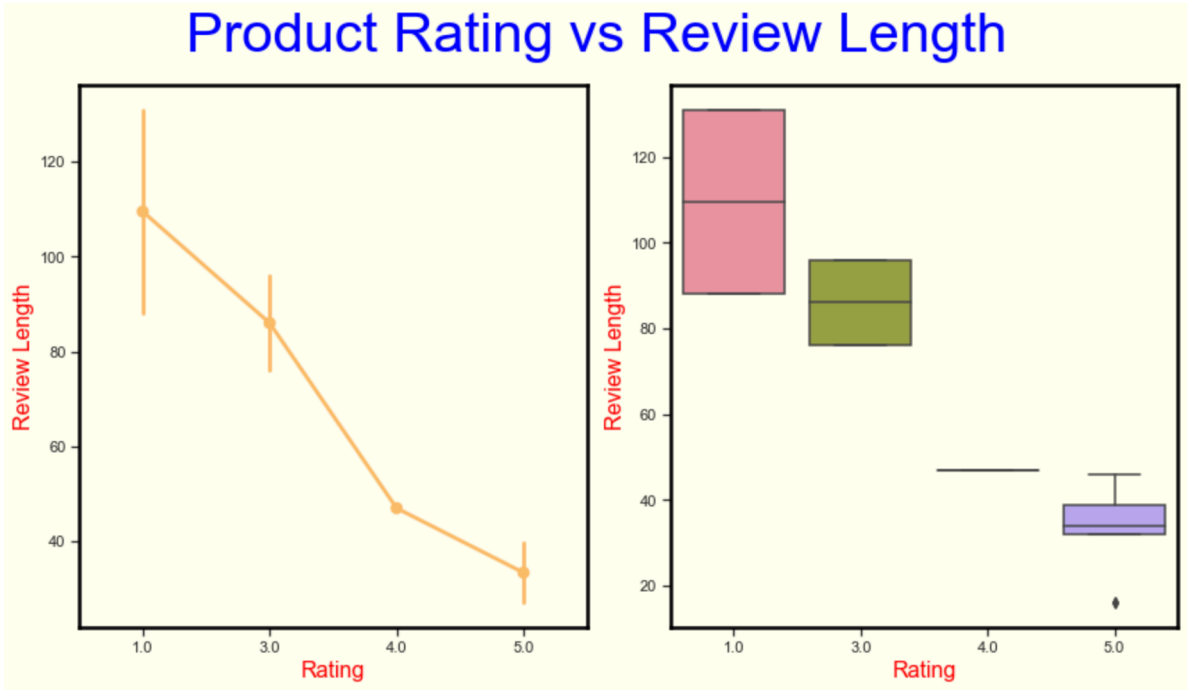
-> Positive Senitment Reviews : 12
-> Neutral Senitment Reviews : 6
-> Neutral Senitment Reviews : 2



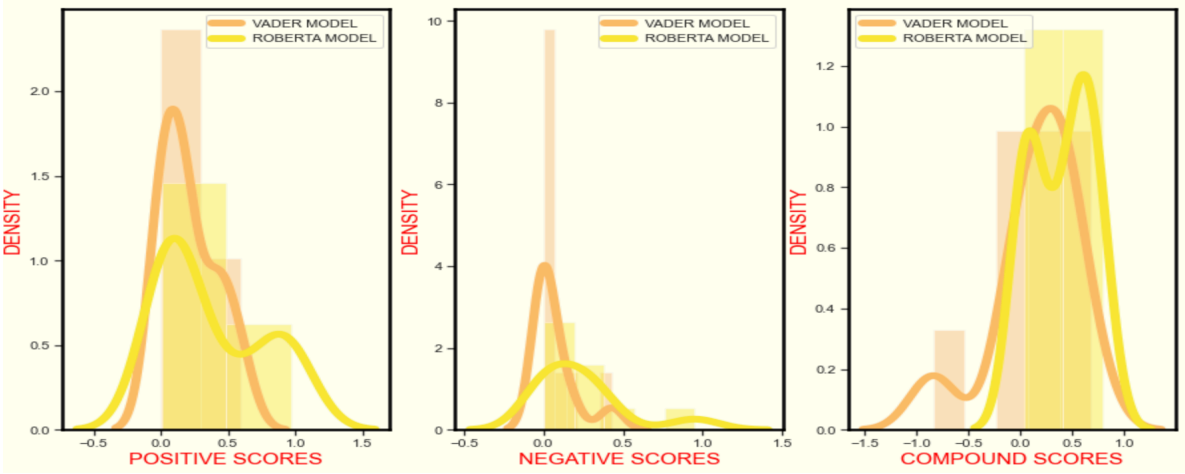

Weighted Sentiment Spread
sentiment
Neutral
Positive
Negative
Density
1.0
0.8
0.6
0.4
0.2
0.0
2021-09
2021-11
2022-01
2022-03
2022-05
2022-07
2022-09
2022-11
2023-01
2023-03
2023-05
Review Date

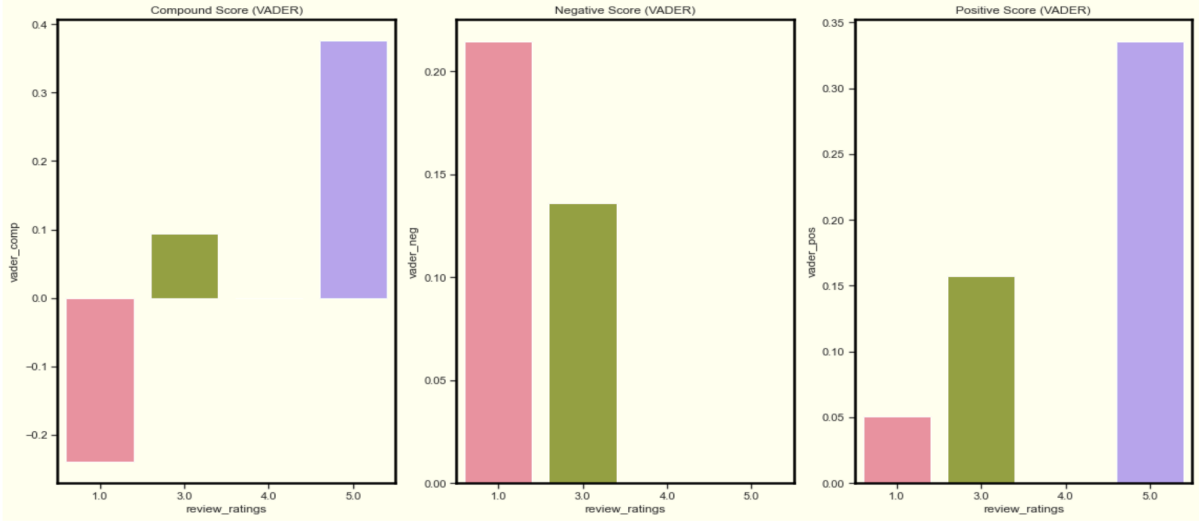

ROBERTA VS VADER MODEL SENTIMENT COMPARISON
Model
VADER
ROBERTA
Review Count
12
10
8
6
4
2
0
Neutral
Positive
Negative
Sentiment

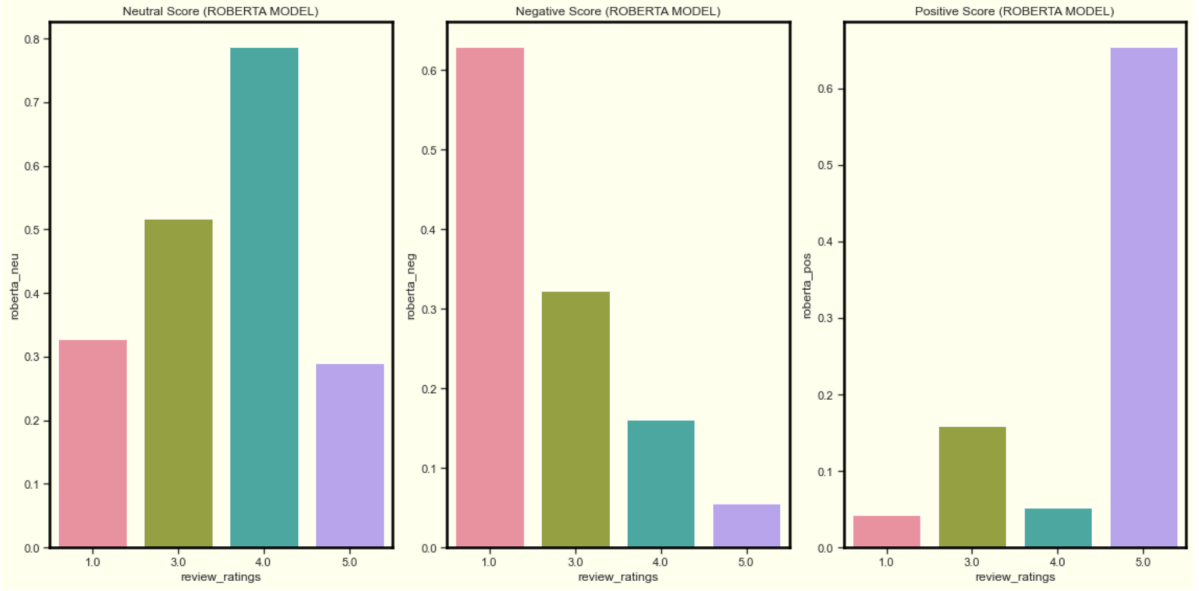

Reviews Count for the product
Review Count
Review Stars
1.0
3.0
4.0
5.0
20%
20%
10%
50%


Product Rating vs Review Length
Review Length
Rating
Review Length
Rating

In-Depth VADER and ROBERTA Model Analysis







[8] Details of Deployment and Presentation

Deployment is in-process, as I have plans of creating a full-fledged website, which is an amazon-like respondent website.

[9] Conclusion and Future Scope

The project was a new revelation but requires more fine touches to make it ready for heavy use. I plan on patenting the web scrapping algorithm and make it more inclusive to other smaller e-commerce platforms.

[10] References

https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/
https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment
https://towardsdatascience.com/sentimental-analysis-using-vader-a3415fef7664
https://www.geeksforgeeks.org/bag-of-words-bow-model-in-nlp/
https://www.zenrows.com/blog/web-scraping-amazon
https://www.datasciencecentral.com/how-to-scrape-amazon-product-data/
https://www.youtube.com/watch?v=QpzMWQvxXWk
https://www.youtube.com/watch?v=HiOtQMcI5wg
https://medium.com/analytics-vidhya/web-scraping-amazon-reviews-a36bdb38b257
https://www.geeksforgeeks.org/web-scraping-amazon-customer-reviews/
https://www.digitalocean.com/community/tutorials/scrape-amazon-product-information-beautiful-soup
https://www.geeksforgeeks.org/scraping-amazon-product-information-using-beautiful-soup/
https://blog.apify.com/step-by-step-guide-to-scraping-amazon/
https://github.com/AlexTheAnalyst/PortfolioProjects/blob/main/Amazon%20Web%20Scraper%20Project.ipynb

Webscrapping.py Module

```python
# Importing required modules/libraries
from bs4 import BeautifulSoup
import requests
import pandas as pd
import numpy as np
import scipy.stats
import seaborn as sns
import random
import matplotlib.pyplot as plt
import warnings
import time
import datetime as dt
import plotly.express as px
import matplotlib.patheffects as pe
import regex as re
import smtplib
import re
from IPython.display import clear_output
warnings.filterwarnings("ignore")

# Finding user agent at: https://httpbin.org/get

# amazon orange, flipkart yellow
colors = ["#FEBD69", "#F8E831"]
customPalette = sns.set_palette(sns.color_palette(colors))

# Change some of seaborn's style settings with `sns.set()`
sns.set(style="ticks",  # The 'ticks' style
        rc={"figure.figsize": (6, 9),  # width = 6, height = 9
            "figure.facecolor": "ivory",  # Figure colour
            "axes.facecolor": "ivory"},
        palette=customPalette)  # Axes colour

# dark black-blue amazon combo, flipkart blue
colors_2 = ["#37475A", "#047BD5"]

# Finding user agent at: https://httpbin.org/get


"""
Amazon Web Scraping Class
"""


class AmazonScraper:

    def __init__(self, productName, pg_no):

        self.user_query = productName  # Storing the searched querry as an attribute

        # Device Specific Headers
        self.head = {
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like
 Gecko) "
                          "Version/16.1 Safari/605.1.15"}

        # Driver Code as attributes (mandatory)
        querry = self.generateUrl(productName, pg_no)  # Getting the required querry
        soup_scrapped = self.connectUrl(querry)  # Connect to the amazon page and scrape it through
BeautifulSoup
        # This will hold all the values of all
        self.cells = soup_scrapped.find_all('div', {'class': 's-result-item',
                                                    'data-component-type': 's-search-result'})

    # Creating a url request based on amazon's website url mechanisms
    def generateUrl(self, query, pg_no):
        query = re.sub(r"\s+", '+', query)
        amazon_url = 'https://www.amazon.in/s?k={0}&page={1}'.format(query, pg_no)
        return amazon_url

    # connecting to the website
    def connectUrl(self, url):
        # s= requests.Session()
        page = requests.get(url, headers=self.head)
        soup = BeautifulSoup(page.content, "html.parser")
        return soup

    # Function to check whether no. of instances haven't exceeded 20
    def lenCheck(self, arr):
```

```
        return False

    #         if len(arr)==20:
    #             return
    #         else:
    #             return False

    # Getter method to retrieve title(name) of each listed product
    def getTitles(self):
        product_titles = []

        for results in self.cells:
            item_name = results.h2.text  # product title (static html --> h2)
            product_titles.append(item_name)

            if self.lenCheck(product_titles):
                break

        return product_titles

        # Getter method to retrieve price(in rs.) of each listed product

    def getPrices(self):
        product_prices = []

        for results in self.cells:
            try:
                item_price = results.find('span', {'class': "a-price-whole"}).text  # (static html --> span tag)
                item_price = float(item_price.replace(",", ""))


            except AttributeError:
                item_price = None

            product_prices.append(item_price)

            if self.lenCheck(product_prices):
                break

        return product_prices

    # Getter method to retrieve review ratings of each listed product
    def getRatings(self):
        product_ratings = []

        for results in self.cells:
            try:
                item_rating = results.find('i', {'class': 'a-icon'}).text

                # Exception, if there are no reviews or ratings avaliable
                try:
                    item_rating = float(item_rating[:3])

                except ValueError:
                    item_rating = 0

            except AttributeError:
                item_rating = 0

            product_ratings.append(item_rating)

            if self.lenCheck(product_ratings):
                break

        return product_ratings

    # Getter method to retrieve number of reviews of each listed product
    def getRatingsCount(self):
        product_ratings_count = []

        for results in self.cells:
            try:
                item_rating_count = results.find('span', {'class': "a-size-base s-underline-text"}).text

                # Exception, if there are no reviews or ratings avaliable
                try:
                    item_rating_count = int(item_rating_count.replace(",", ""))

```

```
                except ValueError:
                    item_rating_count = 0

            except AttributeError:
                item_rating_count = 0

            product_ratings_count.append(item_rating_count)

            if self.lenCheck(product_ratings_count):
                break

        return product_ratings_count

    # Getter method to retrieve images of each listed product
    def getImages(self):
        product_images = []
        for results in self.cells:
            try:
                item_image = results.a.img["src"]

            except TypeError:
                item_image = None

            except AttributeError:
                item_image = None

            product_images.append(item_image)

            if self.lenCheck(product_images):
                break

        return product_images

    # Getter method to retrieve images of each listed product
    def getUrls(self):
        product_urls = []

        for results in self.cells:
            try:
                item_url = "https://www.amazon.in" + results.h2.a["href"]

            except AttributeError:
                item_url = None

            product_urls.append(item_url)

            if self.lenCheck(product_urls):
                break

        return product_urls

    # Converting the scrapped data into an appropriate dataframe
    def toDataframe(self):

        # Printing the array lengths while testing for discrapancies
        """
        print(len(self.getTitles()))
        print(len(self.getUrls()))
        print(len(self.getPrices()))
        print(len(self.getRatings()))
        print(len(self.getRatingsCount()))
        print(len(self.getImages()))
        """

        df = pd.DataFrame({
            "querry_searched": self.user_query,
            "ecommerce_website": "Amazon",
            "product_name": self.getTitles(),
            "product_price": self.getPrices(),
            "product_rating": self.getRatings(),
            "rating_count": self.getRatingsCount(),
            "poduct_image_url": self.getImages(),
            "product_url": self.getUrls()
        })

        return df

    # Additional function to go to each product's url and extract brand name, request response time too
long
```

```
    def addBrand(self, df):
        product_brands = []

        for x in range(len(df)):
            url = df.loc[x][7]
            page = requests.get(url, headers=self.head)
            soup = BeautifulSoup(page.content, "html.parser")

            try:
                product_overview = soup.find(id="productOverview_feature_div")
                cell_detailed = product_overview.find('div', {"class": "a-section a-spacing-small a-spacing
-top-small"})
                cells = cell_detailed.find_all('span')

                y = "Brand"
                for x in range(0, len(cells)):
                    if y in cells[x]:
                        item_brand = cells[x + 1].text


            except AttributeError:
                item_brand = 0

            product_brands.append(item_brand)

        df["product_brand"] = products_brands


"""
Flipkart Web Scraping Class
"""


class FlipkartScraper:

    def __init__(self, productName, pg_no):

        self.grid = False
        self.img = True

        self.user_query = productName  # Storing the searched querry as an attribute

        # Device Specific Headers
        self.head = {
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
like Gecko) "
                          "Version/16.1 Safari/605.1.15"}

        # Driver Code as attributes (mandatory)
        querry = self.generateUrl(productName, pg_no)  # Getting the required querry
        self.soup_scrapped = self.connectUrl(querry)  # Connect to the Flipkart page and scrape it through
BeautifulSoup
        # This will hold all the values of all cells in the page
        self.cells = self.soup_scrapped.find_all('a', {"class": "_1fQZEK"})

        # Flagging self.grid incase the layout of flipkart webpage is gridwise, instead of row wise
        if len(self.cells) == 0:
            self.grid = True
            self.cells = self.soup_scrapped.find_all('div', {"class": "_4ddWXP"})

            if len(self.cells) == 0:
                self.grid = True
                self.img = False
                self.cells = self.soup_scrapped.find_all('div', {"class": "_1xHGtK _373qXS"})


        else:
            self.grid = False

    # Creating a url request based on amazon's website url mechanisms
    def generateUrl(self, query, pg_no):
        query = re.sub(r"\s+", '+', query)
        flipkart_url = 'https://www.flipkart.com/search?q={0}&page={1}'.format(query, pg_no)
        return flipkart_url

    # connecting to the website
    def connectUrl(self, url):
        page = requests.get(url, headers=self.head)
        soup = BeautifulSoup(page.content, "html.parser")
```

```
        return soup

    # Getter method to retrieve title(name) of each listed product
    def getTitles(self):
        product_titles = []

        for index in range(len(self.cells)):
            try:
                if self.grid:
                    item_title = self.cells[index].find_all('a')[1]['title']

                else:
                    item_title = self.cells[index].find('div', {"class": "_4rR01T"}).text

            except AttributeError:
                item_title = 0

            except KeyError:
                item_title = 0

            except ValueError:
                item_title = 0

            product_titles.append(item_title)

        return product_titles

    # Getter method to retrieve price(in rs.) of each listed product
    def getPrices(self):
        product_prices = []
        for index in range(len(self.cells)):
            try:
                if self.grid:
                    str_price = self.cells[index].find_all('a')[2].find('div', {"class": "_30jeq3"}).text
                    item_price = float(str_price[1:].replace(",", ""))

                else:
                    item_price = (self.cells[index].find('div', {"class": "_30jeq3 _1_WHN1"}).text)
                    item_price = float((item_price[1:]).replace(",", ""))

            except AttributeError:
                item_price = None

            product_prices.append(item_price)

        return product_prices

    # Getter method to retrieve review ratings of each listed product
    def getRatings(self):
        product_ratings = []
        for index in range(len(self.cells)):

            try:
                if self.grid:
                    item_rating = float(self.cells[index].find('div', {'class': '_3LWZlK'}).text)

                else:
                    item_rating = float(self.cells[index].find('div', {"class": "_3LWZlK"}).text)

            except AttributeError:
                item_rating = 0

            except ValueError:
                item_rating = 0

            product_ratings.append(item_rating)

        return product_ratings

    # Getter method to retrieve number of ratings of each listed product
    def getRatingsCount(self):
        product_ratings_count = []
        for index in range(len(self.cells)):

            try:

                if self.grid:
                    str_ratings_count = self.cells[index].find('span', {'class': '_2_R_DZ'}).text
                    item_rating_count = int(str_ratings_count.replace("(", "").replace(")", ""))
```

```
                else:
                    ratings_text = self.cells[index].find('span', {"class": "_2_R_DZ"}).text
                    split_position = re.search(" ", ratings_text).start()  # Returns position of first space
                    item_rating_count = int(ratings_text[:split_position].replace(",", ""))


            except ValueError:
                item_rating_count = 0

            except AttributeError:
                item_rating_count = 0

            product_ratings_count.append(item_rating_count)

        return product_ratings_count

    # Getter method to retrieve number of reviews/comments of each listed product
    def getReviewsCount(self):
        product_reviews_count = []
        for index in range(len(self.cells)):
            try:

                if self.grid:
                    item_reviews_count = None
                    # no. of reviews not available in main catalogue grid page

                else:
                    reviews_text = self.cells[index].find('span', {"class": "_2_R_DZ"}).text
                    split_position = re.search(r"\xa0\b", reviews_text).start()
                    item_reviews_count = reviews_text[
                                         split_position + 1:-8]  # we are slicing 8 characters from the end for " Reiews"


            except ValueError:
                item_reviews_count = 0

            except AttributeError:
                item_reviews_count = 0

            product_reviews_count.append(item_reviews_count)

        return product_reviews_count

    # Getter method to retrieve images of each listed product
    def getImages(self):
        product_image_urls = []
        for index in range(0, len(self.cells)):
            try:
                if self.img:
                    item_image_url = self.cells[index].find('img', {'class': '_396cs4'})['src']

                else:
                    item_image_url = self.cells[index].find('img', {'class': '_2r_T1I'})['src']


            except:
                item_image_url = None

            product_image_urls.append(item_image_url)

        return product_image_urls

    # Getter method to retrieve images of each listed product
    def getUrls(self):
        product_urls = []

        for index in range(len(self.cells)):
            try:

                if self.grid:
                    item_qrr = self.cells[index].find_all('a')[1]['href']
                    item_url = "https://www.flipkart.com" + item_qrr

                else:
                    item_url = self.cells[index]['href']
```

```
            except AttributeError:
                item_url = None

            except KeyError:
                item_url = None

            product_urls.append(item_url)

        return product_urls

    # Converting the scrapped data into an appropriate dataframe
    def toDataframe(self):

        # Printing the array lengths while testing for discrapancies
        """
        print(len(self.getTitles()))
        print(len(self.getUrls()))
        print(len(self.getPrices()))
        print(len(self.getRatings()))
        print(len(self.getRatingsCount()))
        print(len(self.getReviewsCount()))
        print(len(self.getImages()))
        """

        df = pd.DataFrame({
            "querry_searched": self.user_query,
            "ecommerce_website": "Flipkart",
            "product_name": self.getTitles(),
            "product_price": self.getPrices(),
            "product_rating": self.getRatings(),
            "rating_count": self.getRatingsCount(),
            "poduct_image_url": self.getImages(),
            "product_url": self.getUrls(),
            "reviews_count": self.getReviewsCount()
        })

        return df


"""
Comments Scraping (Extraction) (Currently Amazon Specific)
"""


class ScrapeComments:

    def __init__(self, df, product_qrd, max_reviews):

        # Device Specific Headers
        self.head = {
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
like Gecko) "
                          "Version/16.1 Safari/605.1.15"}

        self.df_comments = self.getAmazonComments(df, product_qrd, max_reviews)

    # Function that returns the Data Frame created (extracted comments)
    def getDf(self):
        return self.df_comments

    # Main Function that extracts and converts the comments into a structured dataframe
    def getAmazonComments(self, df, onclick_name, max_reviews):

        # Counters
        page_no = 0  # All reviews --> current page number counter
        reviews = 0  # Total reviews extracted counter
        reviews_count = 0  # Counter initialised to prevent item called before assignment error

        # Empty lists that will hold all reviewee data like name, date of review, ratings gien, review
title, content
        customer_name = []
        review_date = []
        ratings = []
        review_title = []
        review_content = []
        # global reviews_soup

        # Finding the url to the queried product name using indexing
```

```
        item_url = list(df['product_url'][df['product_name'] == onclick_name])[0]

        # Connecting (sending a request) to the product's page
        item_page = requests.get(item_url, headers=self.head)
        item_soup = BeautifulSoup(item_page.content, "html.parser")

        try:

            # Till the reviews counter is less than the reviews_count mentioned on the 'all reviews' page
            while reviews <= reviews_count:

                # Finding the link to "all reviews on the product's page", and adding iterating page_no variable
                reviews_url = item_soup.find('a', {'class': 'a-link-emphasis a-text-bold'})['href']
                reviews_url = "https://www.amazon.in" + reviews_url + "&pageNumber=" + str(page_no)

                # Sending request for the Reviews Url page
                reviews_page = requests.get(reviews_url, headers=self.head)
                reviews_soup = BeautifulSoup(reviews_page.content, "html.parser")

                # Number of reviews for the particular page, same throughout all comment (all reviews) pages
                reviews_count = reviews_soup.find('div', {'class': "a-row a-spacing-base a-size-base"}).text
                reviews_count = int(
                    (reviews_count.strip().split('ratings,'))[1].split('with')[0].strip().replace(",", ""))

                # Extracting the required comments in a page
                # These are generally lists containing 10 review related data each.
                names = reviews_soup.select('span.a-profile-name')[2:]
                titles = reviews_soup.select('a.review-title span')
                dates = reviews_soup.select('span.review-date')[2:]
                stars = reviews_soup.select('i.review-rating span.a-icon-alt')[2:]
                content = reviews_soup.select('span.review-text-content span')

                # IndexError Handling : If the sections are left empty by the customers, the index is out of range
                for count in range(len(dates)):

                    try:
                        customer_name.append(names[count].get_text())
                    except IndexError:
                        customer_name.append(0)

                    try:
                        review_date.append(dates[count].get_text().replace("Reviewed in India on ", ""))
                    except IndexError:
                        review_date.append(0)

                    try:
                        review_title.append(titles[count].get_text())
                    except IndexError:
                        review_title.append(None)

                    try:
                        ratings.append(float(stars[count].get_text()[:2]))
                    except IndexError:
                        ratings.append(None)

                    try:
                        review_content.append(content[count].get_text())
                    except IndexError:
                        review_content.append(None)

                    reviews = reviews + 1

                # Incase the number of reviews counter has reached the maximum reviews to be extracted
                if reviews == max_reviews:
                    break

                page_no = page_no + 1
                # print(page_no)


        except:
            pass

        """
        Debugging Section
```

```
        print(len(customer_name))
        print(len(review_date))
        print(len(review_title))
        print(len(ratings))
        print(len(review_content))
        """

        # Recalling required functions from the module.
        df = self.commentsToDf(customer_name, review_date, review_title, ratings, review_content)
        df = self.cleanReviewDates(df)

        return df

    # Function that assigns the lists extracted to a dataframe and returns it.
    def commentsToDf(self, names, dates, titles, ratings, content):

        reviews_df = pd.DataFrame({"customer_name": names,
                                   "review_date": dates,
                                   "review_title": titles,
                                   "review_ratings": ratings,
                                   "review_content": content})

        return reviews_df

    # Function to convert dates which have indian flag and incorrect redundant formats in it, st object in dd/mm/yy
    def cleanReviewDates(self, df):

        cleaned_dates = []
        for comment in list(df['review_date']):
            for string in range(len(comment)):
                if comment[string].isnumeric():
                    cleaned_dates.append(comment[string:])
                    break

        df['review_date'] = cleaned_dates
        df['review_date'] = pd.to_datetime(df['review_date']).dt.strftime('%d/%m/%Y')

        return df


"""
Functions for Creating and merging datasets acccording to a user's querry
"""


# Function that concats 'n' dataframes
def joinDf(df_arr):
    df = pd.DataFrame()
    for x in df_arr:
        df = pd.concat([df, x],
                       ignore_index=True)
    return df


# User Defined Funciton : that checks whether the brand is present in the prodcut_name
def brandCheck(df, brand):
    if len(brand) == 0:
        return df

    else:
        # Creating a regex expression so that if there's upper or lower case in brand or product it'll be accounted
        letters = "("
        for letter in brand:
            upper_x = letter.upper()
            lower_x = letter.lower()
            letters = letters + "[" + upper_x + lower_x + "]"

        letters = letters + "*)\s"
        # print(letters)

        arr_vals = []
        for x in range(0, len(df)):
            product_name = str(df.loc[x][2])
            index_brand = re.search(letters, product_name)
            if index_brand:
                arr_vals.append(df.loc[x])

```

```
        df = pd.DataFrame(arr_vals, columns=list(df.columns))

        return df


# Function that combines the amazon and filpkart webscrapped
def finalDf(amz_df, flk_df, brand_name):
    arr_dfs = []
    arr_dfs.append(amz_df)
    arr_dfs.append(flk_df)
    df1 = joinDf(arr_dfs)
    df = brandCheck(df1, brand_name)
    # df = removePercentile(df,20)
    df.reset_index(inplace=True, drop=True)
    # df.drop(['index'],axis=1,inplace=True)

    return df1, df


# MAIN FUNCTION that takes in the usr query
def execCode(query, brand):
    # Intitla page number
    pg_no = 1
    amz_df_arr = []
    flk_df_arr = []

    # A variable that creates a 100 second buffer between current time and the end.
    t_end = time.time() + 100

    # Until the current sys time isn't less than the end time variable
    while time.time() < t_end:
        try:
            amz_1 = AmazonScraper(query, pg_no)
            flk_1 = FlipkartScraper(query, pg_no)

            amz_df = amz_1.toDataframe()
            flk_df = flk_1.toDataframe()

            amz_df_arr.append(amz_df)
            flk_df_arr.append(flk_df)

            pg_no = pg_no + 1


        except:
            break



    # Joining both the amazon and flipkart pages individually
    amz_df = joinDf(amz_df_arr)
    flk_df = joinDf(flk_df_arr)

    df, brand_present_df = finalDf(amz_df, flk_df, brand)

    return df, brand_present_df

```

ProjectVisualisations.py Module

```python
# Importing required libraries for further analysis
import pandas as pd
import numpy as np
import scipy.stats
import seaborn as sns
import random
import matplotlib.pyplot as plt
import warnings
import time
import plotly.express as px
import matplotlib.patheffects as pe
import regex as re

warnings.filterwarnings("ignore")

# Finding user agent at: https://httpbin.org/get

# amazon orange, flipkart yellow
colors = ["#FEBD69", "#F8E831"]
customPalette = sns.set_palette(sns.color_palette(colors))

# Change some of seaborn's style settings with `sns.set()`
sns.set(style="ticks",  # The 'ticks' style
        rc={"figure.figsize": (6, 9),  # width = 6, height = 9
            "figure.facecolor": "ivory",  # Figure colour
            "axes.facecolor": "ivory"},
        palette=customPalette)  # Axes colour

# dark black-blue amazon combo, flipkart blue
colors_2 = ["#37475A", "#047BD5"]

from webscrapping import AmazonScraper
from webscrapping import FlipkartScraper
from webscrapping import ScrapeComments


# Function to count products with brand name present in comparison to the original web - scrapped dataset
def countActualProductsPlot(df, df_correct):
    #     if len(df)==len(df_correct):
    #         myexplode=[0.5,0]
    #         plt.pie(x = [100,0], explode = myexplode, shadow = True,
    #             colors = ["g","r"])
    #         plt.legend(["Relevant Products","Irrelevant Products"])
    #         plt.title("No brand mentioned / All Products Have Brand Present in Name",
    #                   fontsize=30,color="b")
    #         return plt.show()
    #     else:
    try:
        amazon_prod_count = sum(df["ecommerce_website"] == "Amazon")
        flipkart_prod_count = sum(df["ecommerce_website"] == "Flipkart")

        actual_amazon_prod_count = sum(df_correct["ecommerce_website"] == "Amazon")
        actual_flipkart_prod_count = sum(df_correct["ecommerce_website"] == "Flipkart")

        print("-> Amazon has",
              amazon_prod_count,
              "products listed on each page for the queried product,out of which ",
              (amazon_prod_count - actual_amazon_prod_count),
              " are of other brands")

        print("-> Flipkart has",
              flipkart_prod_count,
              "products listed on each page for the queried product,out of which ",
              actual_flipkart_prod_count,
              " are of other brands")

        plt.figure(figsize=(12, 7))
        plt.rcParams["axes.edgecolor"] = "black"
        plt.rcParams["axes.linewidth"] = 2.50
        plt.suptitle("Stacked Bar Chart Comparing Relevance of Listed Products", fontsize=30, color='b')

        # Bar Plot
        plt.subplot(1, 2, 1)
        x = ["Amazon", "Flipkart"]
        y1 = [amazon_prod_count - actual_amazon_prod_count,
              flipkart_prod_count - actual_flipkart_prod_count]

        y2 = [actual_amazon_prod_count, actual_flipkart_prod_count]

```

```
        # plot bars in stack manner
        plt.bar(x, y1, color='red', edgecolor=colors_2)
        plt.bar(x, y2, bottom=y1, color=colors, edgecolor=colors_2)

        # Labels
        plt.xlabel("E-Commerce Website", fontsize=16, color="red")
        plt.ylabel("Count of Products", fontsize=16, color="red")
        plt.legend(["Irrelevant Products", "Relevant Products Amazon", "Relevant Products Flipkart"])

        # Pie Chart
        plt.subplot(1, 2, 2)
        myexplode = [0.4, 0]

        relv_prd = sum(y2) / (amazon_prod_count + flipkart_prod_count)
        irrelv_prd = 1 - relv_prd

        plt.pie(x=[relv_prd, irrelv_prd], explode=myexplode, shadow=True,
                colors=["g", "r"])
        plt.legend(["Relevant Products", "Irrelevant Products"])
        plt.tight_layout()
        return plt.show()

    except:
        print("Error !")


def countProductsPlot(df):
    amazon_prod_count = sum(df["ecommerce_website"]=="Amazon")
    flipkart_prod_count = sum(df["ecommerce_website"] == "Flipkart")

    print("-> Amazon has", amazon_prod_count, "products listed on each page.")
    print("-> Flipkart has", flipkart_prod_count, "products listed on each page.")

    plt.figure(figsize=(12, 7))
    plt.rcParams["axes.edgecolor"] = "black"
    plt.rcParams["axes.linewidth"] = 2.50
    plt.suptitle("Products Listed Per Page for the Query", fontsize=30, color='b')
    plt.subplot(1, 2, 1)
    # fig.subtitle('Products per page for given product')

    sns.countplot(x='ecommerce_website', data=df, color=customPalette,
                  linewidth=3, edgecolor=colors_2)

    plt.xlabel("Ecommerce Website", fontsize=16, color="red")
    plt.ylabel("No. of Products per page", fontsize=16, color="red")

    plt.subplot(1, 2, 2)
    plt.pie(x=[amazon_prod_count,flipkart_prod_count], labels=["Amazon", "Flipkart"],
            colors=customPalette, autopct='%.0f%%')

    plt.tight_layout()
    return plt.show()


# Function to find average of a clolumn for the particular e-commerce website
def subset_avg_gen(df, website, col):
    df_subset = df[df["ecommerce_website"] == website]
    average = df_subset[col].mean()
    return average


# User Defined Function : Average Price of the products displayed upon query, website wise
def priceCompLine(df):
    x = list(df[df["ecommerce_website"] == "Amazon"]["product_price"])
    y = list(df[df["ecommerce_website"] == "Flipkart"]["product_price"])
    plt.figure(figsize=(15, 10))
    plt.rcParams["axes.edgecolor"] = "black"
    plt.rcParams["axes.linewidth"] = 2.50
    overlapping = 0.1

    plt.plot(x, lw=6, path_effects=[pe.SimpleLineShadow(shadow_color=colors_2[0]), pe.Normal()])
    plt.plot(y, lw=6, path_effects=[pe.SimpleLineShadow(shadow_color=colors_2[1]), pe.Normal(), ])

    plt.legend(["Amazon", "Flipkart"], loc='upper left')
    plt.xlabel("Product No.", color="red")
    plt.ylabel("Price (in rs.)", color="red")
    plt.title("Comparing Prices (line-plot)", fontsize=30, c='b')
    return plt.show()
```

```
159
160
161 def CompPriceBox(df):
162     amz_avg_price = subset_avg_gen(df, "Amazon", "product_price")
163     flk_avg_price = subset_avg_gen(df, "Flipkart", "product_price")
164
165     print("Average Product Price for queried product at amazon is : ", amz_avg_price)
166     print("Average Product Price for queried product at flipkart is : ", flk_avg_price)
167
168     plt.rcParams["axes.edgecolor"] = "black"
169     plt.rcParams["axes.linewidth"] = 2.50
170     plt.figure(figsize=(15, 10))
171     # Box plot
172     b = sns.boxplot(data=df,
173                     x="ecommerce_website",
174                     y="product_price",
175                     width=0.4,
176                     linewidth=2,
177
178                     showfliers=False)
179     # Strip plot
180
181     b = sns.stripplot(data=df,
182                       x="ecommerce_website",
183                       y="product_price",
184                       # Colours the dots
185                       linewidth=1,
186                       alpha=0.4)
187
188     b.set_ylabel("Price", fontsize=14, color="red")
189     b.set_xlabel("E-Commerce Website", fontsize=14, color="red")
190     b.set_title("Comparing Prices (box-plot)", fontsize=30, c='b')
191
192     sns.despine(offset=5, trim=True)
193
194     return b.get_figure()
195
196
197 # User Defined Function : Number of products available on Amazon and flipkart for the given querry
198 def priceNamePlotly(df):
199     fig = px.line(df, x="product_name", y="product_price",
200                   color="ecommerce_website", template="plotly_dark",
201                   color_discrete_sequence=colors,
202                   title="Products and their price")
203
204     fig.update_xaxes(showticklabels=False)
205     return fig.show()
206
207
208 # User Defined Function : Average Rating of the products displayed upon query, website wise
209 def CompRatingBox(df):
210     amz_avg_ratings = subset_avg_gen(df, "Amazon", "product_rating")
211     flk_avg_ratings = subset_avg_gen(df, "Flipkart", "product_rating")
212
213     print("Average Product Rating for queried product at amazon is : ", amz_avg_ratings)
214     print("Average Product Rating for queried product at flipkart is : ", flk_avg_ratings)
215
216     plt.figure(figsize=(15, 10))
217
218     # Box plot
219     b = sns.boxplot(data=df,
220                     x="ecommerce_website",
221                     y="product_rating",
222                     width=0.4,
223                     linewidth=2,
224
225                     showfliers=False)
226     # Strip plot
227     b = sns.stripplot(data=df,
228                       x="ecommerce_website",
229                       y="product_rating",
230                       # Colours the dots
231                       linewidth=1,
232                       alpha=0.4)
233
234     b.set_ylabel("Ratings (out of 5)", fontsize=14, color="red")
235     b.set_xlabel("E-Commerce Website", fontsize=14, color="red")
236     b.set_title("Comparing Ratings (box-plot)", fontsize=30, c='b')
237
```

```
    sns.despine(offset=5, trim=True)
    # b.get_figure()
    return b


# User Defined Function : Analysing the relationship between Product's Price and its Rating
# Function that plots a scatter plot between price and ratings
def price_rat_corr(df):
    # Printing what kind of correlation is present between product price and product rating
    price_rating_corr = df.corr()['product_rating']["product_price"]

    if abs(price_rating_corr) > 0.9:
        print("Product Ratings and Product Prices are very hignly correlated. ")

    if abs(price_rating_corr) > 0.7:
        print("Product Ratings and Product Prices are hignly correlated. ")

    if abs(price_rating_corr) > 0.5:
        print("Product Ratings and Product Prices are moderately correlated. ")

    if abs(price_rating_corr) > 0.3:
        print("Product Ratings and Product Prices have low correlation. ")

    else:
        print("Product Ratings and Product Prices have very little to no correlation. ")

    fig = px.scatter(df, x="product_rating", y="product_price",
                     color="ecommerce_website", template="plotly_dark",
                     color_discrete_sequence=colors,
                     trendline="ols",
                     title="Analysing the relationship between Product's Price and Product's Ratings")

    fig.update_xaxes(showticklabels=True)
    return fig.show()


"""
Main function for vsiualising all the results
"""


def visualiseQueryReults(df, brand_present_df):
    # amazon orange, flipkart yellow
    colors = ["#FEBD69", "#F8E831"]
    customPalette = sns.set_palette(sns.color_palette(colors))

    # Change some of seaborn's style settings with `sns.set()`
    sns.set(style="ticks",  # The 'ticks' style
            rc={ "figure.facecolor": "ivory",  # Figure colour
                "axes.facecolor": "ivory"},
            palette=customPalette)  # Axes colour

    ax1 = countProductsPlot(df)
    if len(brand_present_df)==len(df):
        pass
    else:
        ax2 = countActualProductsPlot(df, brand_present_df)

    ax3 = priceCompLine(brand_present_df)
    ax4 = CompPriceBox(brand_present_df)
    ax5 = priceNamePlotly(brand_present_df)

    ax6 = CompRatingBox(brand_present_df)

    ax7 = price_rat_corr(brand_present_df)

```

ProductReviewAnalysis.py Module

```python
# Importing Required libraries and modules

from bs4 import BeautifulSoup
import requests
import time
import datetime

import smtplib
import re

import pandas as pd
import seaborn as sns
import datetime as dt
import numpy as np
# Finding user agent at: https://httpbin.org/get

from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm

from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification
from scipy.special import softmax
from IPython.display import clear_output
from tabulate import tabulate
import urllib.request
# from PIL import Image
from IPython.display import Image
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt

stopwords = set(STOPWORDS)

# Change some of seaborn's style settings with `sns.set()`
sns.set(style="ticks",  # The 'ticks' style
        rc={"figure.figsize": (6, 9),  # width = 6, height = 9
            "figure.facecolor": "ivory",  # Figure colour
            "axes.facecolor": "ivory"})  # Axes colour

# Importing files created by me, and saved in. webscrapping.py
from webscrapping import AmazonScraper
from webscrapping import FlipkartScraper
from webscrapping import ScrapeComments

MODEL = f"cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)

class color:
    PURPLE = '\033[95m'
    CYAN = '\033[96m'
    DARKCYAN = '\033[36m'
    BLUE = '\033[94m'
    GREEN = '\033[92m'
    YELLOW = '\033[93m'
    RED = '\033[91m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'
    END = '\033[0m'


"""
User Defined Functions
"""


# User Defined Function To neatly print the products of the particular dataset
def printTableProducts(df, length):
    products_dict = dict(df[df["ecommerce_website"] == "Amazon"]["product_name"])
    arr = []

    for x in products_dict:
        arr.append([x, products_dict.get(x)[:70]])

        if len(arr) == length:
            break

    print(tabulate(arr, headers=['Product No.', 'Product Name']))
```

```python


# User Defined Function To Show the selected products details in a neat manner
def showProductDeets(df, index):
    element = df.iloc[index]

    print((color.BOLD + color.BLUE + 'PRODUCT DETAILS' + color.END).center(100, "-"))
    print()
    print(color.BOLD + 'Product Name : ' + color.END, element[2])
    print(color.BOLD + 'Product Price : ' + color.END, element[3])
    print(color.BOLD + 'Average Product Rating : ' + color.END, element[4])
    print(color.BOLD + 'Total Number of Ratings : ' + color.END, element[5])
    print()

    urllib.request.urlretrieve(df.iloc[index][6], "ex.png")
    display(Image(filename='ex.png'))
    print("-" * 100)


# Function to add a new column that contains the size of each review
def addReviewLen(df):
    len_review = []
    for x in range(len(df)):
        rev_size = len(df.iloc[x][4])
        len_review.append(rev_size)
    df['review_size'] = len_review


# Function to clean the given comments dataset (drop na/null values)
def cleanCommentsDf(df):
    df.dropna(inplace=True)
    df.reset_index(inplace=True, drop=True)


# Return Mapped value based on Categorical variables for weighted calculations
def mapValue(element):
    # 'Positive', 'Neutral', 'Negative' unqiue values
    if element == 'Positive':
        return 1

    if element == 'Neutral':
        return 0

    else:
        return -1


"""
VADER AND ROBERTA MODEL related user defined functions
"""


# Vader model (applying vader model on each comment and creating a df)
def getVaderDf(df):
    global sia
    sia = SentimentIntensityAnalyzer()
    # Runn the polarity score on the entire dataset
    res = {}
    for i, row in tqdm(df.iterrows(), total=len(df), desc="Vader Model"):
        text = row['review_content']
        index = list(df.index)[i]
        val = sia.polarity_scores(text)
        res[index] = {
            'vader_neg': float(val['neg']),
            'vader_pos': float(val['pos']),
            'vader_comp': float(val['compound'])
        }

    # Converting thed dictionary into a dataframe and using transpose to make columns as rows
    vaders = pd.DataFrame(res).T
    vaders.reset_index(inplace=True, drop=True)
    # Merfing the vaders dataframe with the original dataset

    return vaders


# Roberta Model Scores (individual)
def polarity_scores_roberta(example):
    if len(example)>2150:
```

```
        example = example[:2000]
    #if len(example.split()) > 500:
        #example = " ".join(example.split()[:500])

    encoded_text = tokenizer(example, return_tensors='pt')
    output = model(**encoded_text)
    scores = output[0][0].detach().numpy()
    scores = softmax(scores)
    scores_dict = {
        'roberta_neg': scores[0],
        'roberta_neu': scores[1],
        'roberta_pos': scores[2]
    }
    return scores_dict


# Roberta model (applying Roberta model on each comment and creating a df)
def getRobetaDf(df):

    # Runn the polarity score on the entire dataset
    res = {}
    for i, row in tqdm(df.iterrows(), total=len(df), desc="Roberta Model"):
        text = row['review_content']
        index = list(df.index)[i]
        val = polarity_scores_roberta(text)
        res[index] = val

    # Converting thed dictionary into a dataframe and using transpose to make columns as rows
    df = pd.DataFrame(res).T
    df.reset_index(inplace=True, drop=True)

    return df


# User defined function to create a categorical column on the basis of the three Vader values
def getVaderSentiment(df):
    arr = []
    for x in range(len(df)):
        vader_compound_score = df["vader_comp"][x]

        if vader_compound_score >= 0.05:
            arr.append("Positive")

        elif vader_compound_score <= - 0.05:
            arr.append("Negative")

        else:
            arr.append("Neutral")

    if 'vader_sentiment' not in df.columns:
        df.insert(loc=6,
                  column='vader_sentiment',
                  value=arr)

    else:
        df['vader_sentiment'] = arr


# User defined function to create a categorical column on the basis of the three Roberta models
def getRobertaSentiment(df):
    arr = []
    for pos in range(len(df)):
        roberta_sentiment_dict = {"Positive": df[["roberta_pos", "roberta_neg", "roberta_neu"]].iloc[pos][0
],
                                  "Negative": df[["roberta_pos", "roberta_neg", "roberta_neu"]].iloc[pos][1
],
                                  "Neutral": df[["roberta_pos", "roberta_neg", "roberta_neu"]].iloc[pos][2
]}

        roberta_sentiment_value = max(roberta_sentiment_dict, key=roberta_sentiment_dict.get)

        arr.append(roberta_sentiment_value)

    if 'roberta_sentiment' not in df.columns:
        df.insert(loc=6,
                  column='roberta_sentiment',
                  value=arr)

    else:
```

```
        df['roberta_sentiment'] = arr


# User defined function : Creating a weighted column based on
def getFinalSentiment(df):
    arr = []
    roberta_sentiment_vals = list(map(mapValue, list(df['roberta_sentiment'])))
    vader_sentiment_vals = list(map(mapValue, list(df['vader_sentiment'])))

    roberta_val = list(df['roberta_sentiment'])
    vader_val = list(df['vader_sentiment'])

    for x in range(len(df)):
        val = float((roberta_sentiment_vals[x] * 0.6) + (vader_sentiment_vals[x] * 0.4))

        if val >= 0.19:
            arr.append("Positive")

        elif val >= (-0.4):
            arr.append("Neutral")


        elif val >= (-1):
            arr.append("Negative")

    if 'sentiment' not in df.columns:
        df.insert(loc=6,
                  column='sentiment',
                  value=arr)

    else:
        df['sentiment'] = arr


"""
Plotting related user defined functions
"""


# Function for plotting a comparison on the kinds of reviews given for the product
def ratingsItemPlot(df):
    plt.figure(figsize=(12, 7))
    plt.subplot(1, 2, 1)

    rev_count = list(df['review_ratings'].value_counts().sort_index())
    rev_tags = list(df['review_ratings'].unique())
    rev_tags.sort()

    plt.bar(rev_tags, rev_count)

    plt.plot(rev_tags, rev_count, color='black', linewidth=7.0)

    plt.xlabel('Review Stars', fontsize=16, color="red")
    plt.ylabel('Review Count', fontsize=16, color="red")

    plt.suptitle("Reviews Count for the product", fontsize=30, color='b')

    plt.subplot(1, 2, 2)
    #myexplode = [0.15, 0.19, 0.22, 0.12, 0.15]

    plt.pie(x=rev_count,
            shadow=True, autopct='%.0f%%')

    plt.legend(rev_tags)
    # plt.rcParams.update({'font.size': 10})

    plt.tight_layout()
    return plt.show()


# User defined function to count and show a woordle like frequency chart
def show_wordcloud(data):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        max_words=200,
        max_font_size=40,
        scale=3,
        random_state=1  # chosen at random by flipping a coin; it was heads
```

```
    ).generate(str(data))

    fig = plt.figure(1, figsize=(15, 15))
    plt.axis('off')

    plt.imshow(wordcloud)
    return plt.show()


# Graph that compares the review length and the ratings given
def sizeComPlot(df):
    plt.figure(figsize=(12, 7))
    plt.suptitle("Product Rating vs Review Length", fontsize=40, color="blue")
    plt.subplot(1, 2, 1)
    sns.pointplot(data=df, x="review_ratings", y="review_size")
    plt.xlabel("Rating", fontsize=16, color="red")
    plt.ylabel("Review Length", fontsize=16, color="red")

    plt.subplot(1, 2, 2)
    sns.boxplot(data=df, x="review_ratings", y="review_size")
    plt.xlabel("Rating", fontsize=16, color="red")
    plt.ylabel("Review Length", fontsize=16, color="red")
    plt.tight_layout()
    return plt.show()


# User defined function to compare Vader and Roberta Model through dist plot
def compModelsGraph(df):
    plt.figure(figsize=(15, 8))
    plt.suptitle("In-Depth VADER and ROBERTA Model Analysis", color="blue", fontsize=30)

    plt.subplot(1, 3, 1)
    plot_one = sns.distplot(list(df['vader_pos']), kde_kws=dict(linewidth=7))
    plot_two = sns.distplot(df['roberta_pos'], kde_kws=dict(linewidth=7))
    plt.xlabel("POSITIVE SCORES", color='red', fontsize=16)
    plt.ylabel("DENSITY", color='red', fontsize=16)
    plt.legend(["VADER MODEL", "ROBERTA MODEL"])

    plt.subplot(1, 3, 2)
    plot_one = sns.distplot(list(df['vader_neg']), kde_kws=dict(linewidth=7))
    plot_two = sns.distplot(df['roberta_neg'], kde_kws=dict(linewidth=7))
    plt.xlabel("NEGATIVE SCORES", color='red', fontsize=16)
    plt.ylabel("DENSITY", color='red', fontsize=16)
    plt.legend(["VADER MODEL", "ROBERTA MODEL"])

    plt.subplot(1, 3, 3)
    plot_one = sns.distplot(list(df['vader_comp']), kde_kws=dict(linewidth=7))
    plot_two = sns.distplot(df['roberta_neu'], kde_kws=dict(linewidth=7))
    plt.xlabel("COMPOUND SCORES", color='red', fontsize=16)
    plt.ylabel("DENSITY", color='red', fontsize=16)
    plt.legend(["VADER MODEL", "ROBERTA MODEL"])

    return plt.show()


# Plotting Vader Scores vs Ratings
def plotVaderResults(df):
    fig, axs = plt.subplots(1, 3, figsize=(16, 8))
    sns.barplot(data=df, x='review_ratings', y='vader_comp', ax=axs[0], ci=None)
    axs[0].set_title("Compound Score (VADER)")

    sns.barplot(data=df, x='review_ratings', y='vader_neg', ax=axs[1], ci=None)
    axs[1].set_title("Negative Score (VADER)")

    sns.barplot(data=df, x='review_ratings', y='vader_pos', ax=axs[2], ci=None)
    axs[2].set_title("Positive Score (VADER)")

    plt.tight_layout()
    return plt.show()


# Plotting Roberta Model Scores vs Ratings
def plotRobertaResults(df):
    fig, axs = plt.subplots(1, 3, figsize=(16, 8))
    sns.barplot(data=df, x=df['review_ratings'], y=df['roberta_neu'], ax=axs[0], ci=None)
    axs[0].set_title("Neutral Score (ROBERTA MODEL)")

    sns.barplot(data=df, x=df['review_ratings'], y=df['roberta_neg'], ax=axs[1], ci=None)
    axs[1].set_title("Negative Score (ROBERTA MODEL)")
```

```
393
394     sns.barplot(data=df, x=df['review_ratings'], y=df['roberta_pos'], ax=axs[2], ci=None)
395     axs[2].set_title("Positive Score (ROBERTA MODEL)")
396
397     plt.tight_layout()
398     return plt.show()
399
400
401 # Density Plot for final sentiment density (Date wise)
402 def sentimentDensityPlot(df):
403     plt.figure(figsize=(12, 7))
404     sns.kdeplot(data=df, x='review_date', hue="sentiment", palette="icefire",
405                 multiple="fill")
406     plt.title("Weighted Sentiment Spread", fontsize=30, color='b')
407     plt.xlabel("Review Date", fontsize=16, color="red")
408     plt.ylabel("Density", fontsize=16, color="red")
409     return plt.show()
410
411
412 # Bar Chart to compare the Sentiment values of ROBERTA and VADER model
413 def modelValsComp(df):
414     rob_vad_vals = list(df["vader_sentiment"]) + list(df["roberta_sentiment"])
415     which_vals = ["VADER" for x in range(len(df))] + ["ROBERTA" for x in range(len(df))]
416     temp_df = pd.DataFrame({"Sentiment": rob_vad_vals, "Model": which_vals})
417
418     plt.figure(figsize=(12, 7))
419     sns.histplot(data=temp_df, x="Sentiment", hue="Model", multiple="dodge", shrink=.8)
420     plt.title("ROBERTA VS VADER MODEL SENTIMENT COMPARISON", fontsize=30, color='b')
421     plt.xlabel("Sentiment", fontsize=16, color="red")
422     plt.ylabel("Review Count", fontsize=16, color="red")
423     return plt.show()
424
425
426 # Bar Chart and Pie Chart of weighted Sentiment Values
427 def sentimentPlot(df):
428     pos_count = sum(df["sentiment"] == "Positive")
429     neu_count = sum(df["sentiment"] == "Neutral")
430     neg_count = sum(df["sentiment"] == "Negative")
431
432     print("-> Positive Senitment Reviews : ", pos_count)
433     print("-> Neutral Senitment Reviews : ", neu_count)
434     print("-> Neutral Senitment Reviews : ", neg_count)
435
436     plt.figure(figsize=(12, 7))
437     plt.rcParams["axes.edgecolor"] = "black"
438     plt.rcParams["axes.linewidth"] = 2.50
439     plt.suptitle("Reviews Sentiment Chart (Weighted 3:2 (ROBERTA MODEL : VADER MODEL))", fontsize=30, color
    ='b')
440     plt.subplot(1, 2, 1)
441
442     plt.bar(x=["Positive", "Neutral", "Negative"],
443             height=[pos_count, neu_count, neg_count],
444             color=["green", 'yellow', 'red'])
445
446     plt.xlabel("Sentiment", fontsize=16, color="red")
447     plt.ylabel("Count of reviews", fontsize=16, color="red")
448
449     plt.subplot(1, 2, 2)
450     plt.pie(x=[pos_count, neu_count, neg_count], labels=["Positive", "Neutral", "Negative"],
451             colors=['green', 'yellow', 'red'], autopct='%.0f%%')
452
453     plt.tight_layout()
454     return plt.show()
455
456
457 """
458 MAIN FUNCTION
459 """
460
461
462 def mainReviiewsAnalysis(df, product_name, cmts_size):
463     # Creating a subset of only Amazon related products
464     amz_df = df[df["ecommerce_website"] == "Amazon"]
465
466     # Scrapping the amazon product's review using the module created by me in Webscrapping.py
467     comments_scrapping = ScrapeComments(amz_df, product_name, cmts_size)
468     df = comments_scrapping.getDf()
469
470     # Checking once more for null/na/redundant values in the freshly scraped df
```

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

    cleanCommentsDf(df)
    addReviewLen(df)
    # Using the user defined function to generate vader and roberta model values for the comments
    vader_df = getVaderDf(df)
    roberta_df = getRobetaDf(df)

    # Joining the initial data
    x = roberta_df.join(vader_df)
    df_cmt = df.join(x)
    getVaderSentiment(df_cmt)
    getRobertaSentiment(df_cmt)
    getFinalSentiment(df_cmt)

    return df_cmt


# Visualising Plots function with the help of the otehr methods created
def visualiseCommentsAnalysis(df_cmts):
    plt0 = show_wordcloud(list(df_cmts['review_content']))
    plt1 = sentimentPlot(df_cmts)
    plt2 = sentimentDensityPlot(df_cmts)
    plt3 = modelValsComp(df_cmts)
    plt4 = ratingsItemPlot(df_cmts)
    plt5 = sizeComPlot(df_cmts)
    plt6 = compModelsGraph(df_cmts)
    plt7 = plotVaderResults(df_cmts)
    plt8 = plotRobertaResults(df_cmts)

```