**Flipkart Live Data Product Recommendation System With Sentimental Analysis**

**Project Report**

**Submitted by**

*Balamurugan Arunachalam*

**Table of Contents**

1) Project Overview

1.1) Introduction

1.2) Skills and Technologies Used

1.3) Problem Statement

2) Technical Approach

2.1) Web Scraping using Selenium & BeautifulSoup

2.2) Data Cleaning and Preprocessing

2.3) Sentimental Analysis

2.4) Data Visualization

2.5) Langchain Implementation

2.6) AWS Deployment

3) Web Scraping Process

3.1) Technologies Used

3.2) Project Structure

3.2.1) Initialization

3.2.2) Reading Product Links

3.2.3) Scraping Function (scrape\_product\_details)

3.2.4) Saving Scraped Data

3.2.5) Driver Cleanup

3.3) Future Improvements

4) Data Cleaning and Preprocessing

4.1) Libraries Used

4.2) Steps for Preprocessing the Data

4.2.1) Loading the Required Libraries

4.2.2) Loading the Dataset

4.2.3) Data Cleaning

4.2.4) Tokenization, Stopword Removal, and Lemmatization

4.2.5) Handling Numeric Fields

4.2.6) Final Steps

5) Sentimental Analysis

5.1) Libraries Used

5.2) Steps for Sentimental Analysis

5.2.1) Loading the Preprocessed Data

5.2.2) TextBlob Sentiment Analysis

5.2.3) Applying Sentiment Analysis:

5.2.4) Visualization

5.2.5) Product-Level Sentiment Summary

5.2.6) Saving Sentiment Summary

6) Data Visualization

6.1) Libraries Used

6.2) Steps for Visualization

6.2.1) Rating Distribution

6.2.2) Price Distribution

6.2.3) Top 10 Products by Average Rating

7) LangChain & Streamlit Implementation

7.1) Libraries and Setup

7.2) Loading Sentiment Data:

7.3) Initializing Cohere API

7.4) Product Recommendation Logic:

7.5) Streamlit UI Setup:

7.5.1) Page Config and Header:

7.5.2) User Input Section

7.5.3) Recommendation Display

7.5.4) Footer

8) Conclusion

**1)Project Overview :**

**1.1) Introduction :**

This project focuses on scraping product reviews from Flipkart, analyzing customer sentiment, and providing data-driven product recommendations. By utilizing natural language processing (NLP) techniques such as sentiment analysis, the project helps enhance the product recommendation engine based on customer reviews.

**1.2) Skills and Technologies Used :**

* **Web Scraping**: Selenium, Beautiful Soup
* **Data Cleaning and Structuring**: Pandas
* **Sentiment Analysis**: TextBlob
* **Data Visualization**: Matplotlib, Seaborn
* **Product Recommendation**: LangChain
* **Deployment**: Streamlit, AWS

**1.3) Problem Statement :**

The problem is to develop an automated system that scrapes customer reviews from Flipkart, processes them using natural language processing (NLP) techniques to analyze sentiment, and provides product recommendations based on user feedback.

**2)Technical Approach**

**2.1) Web Scraping using Selenium & BeautifulSoup :**

* + We use **Selenium** to automate the browser and extract product review data (Product ID, Review Text, Rating) from Flipkart. We first fetch product URLs and then scrape the reviews for each product.
  + **Beautiful Soup** helps in parsing the HTML content to extract product reviews.

**2.2) Data Cleaning and Preprocessing :**

Once the reviews are collected, we clean the data by removing duplicates, special characters, and stopwords. We also normalize the text to ensure consistent formatting for sentiment analysis.

**2.3) Sentimental Analysis :**

The sentiment of each review is determined using the TextBlob library. TextBlob classifies each review as positive, negative, or neutral based on the review's polarity.

**2.4) Data Visualization :**

We created visualizations to display the sentiment distribution and top recommended products.

**2.5) Langchain Implementation :**

LangChain is used to build a recommendation system based on the sentiment analysis data. Products with the highest number of positive reviews are recommended.

**2.6) AWS Deployment :**

We deploy the recommendation engine using Streamlit and host it on AWS. The app allows users to input preferences and get real-time recommendations based on customer sentiment.

**3) Web Scraping Process :**

This project utilizes Selenium for automating web browser interactions and BeautifulSoup for parsing the HTML structure of the pages. The process consists of three main components: setting up Selenium, scraping product details (name, price), and extracting reviews with sentiment analysis.

**3.1) Technologies Used:**

* **Python**: The primary programming language used for the scraping code.
* **Selenium**: A web automation tool used for controlling the web browser to navigate pages, click buttons, and retrieve page content.
* **Beautiful** **Soup**: A Python library for parsing HTML and XML documents and extracting data from web pages.
* **CSV**: A module for reading from and writing to CSV files, which is used to store the scraped data.

**3.2) Project Structure**

**3.2.1) Initialization:**

* Set up the Selenium WebDriver (Chrome in this case) to automate browsing.
* Define file paths for reading input links and writing output data.

**3.2.2) Reading Product Links:**

* Load product links from a CSV file (Cleaned\_Phone\_Links.csv). Each link points to an individual product page on Flipkart.

**3.2.3) Scraping Function (scrape\_product\_details):**

The main function that performs the scraping process for a given product link.

* **Product Details Extraction**:
  + Uses CSS selectors to locate and extract the product name and price from the product page.
* **Review Section Access:**
  + Clicks on the "All Reviews" button to access the reviews section of the product page.
* **Review Data Scraping:**
  + Loops through multiple pages of reviews (up to a maximum of 25 pages).
  + For each review, it extracts the rating and the full review text.
  + It handles "Read More" buttons to ensure complete review text is captured.
* **Pagination Handling:**
  + Checks for a "Next" button to navigate through additional pages of reviews.

**3.2.4) Saving Scraped Data:**

After scraping data for a product, the results (product link, name, price, rating, and review) are written to an output CSV file (Mobile\_Phone\_Data.csv).

**3.2.5) Driver Cleanup**

Quits the browser after all products have been processed to free resources.

**3.3) Future Improvements:**

* Implement more sophisticated error logging.
* Optimize waiting mechanisms using conditions rather than fixed sleep times.
* Explore the use of proxies or user-agent rotation to handle potential rate limits imposed by the website.

**4) Data Cleaning and Preprocessing**

**4.1) Libraries Used:**

* **pandas**: To load, manipulate, and clean the dataset.
* **re**: Regular expressions for text cleaning.
* **spacy**: For tokenization, stopword removal, and lemmatization.
* **nltk**: For downloading and using the stopwords list.

**4.2) Steps for Preprocessing the Data :**

**4.2.1) Loading the Required Libraries :**

* The necessary libraries, including pandas, regular expressions (re), spaCy, and NLTK, are imported.
* The stopwords are loaded using NLTK, and if required, the en\_core\_web\_sm model for spaCy is downloaded.

import pandas as pd

import re

import spacy

import nltk

from nltk.*corpus* import stopwords # Import stopwords here

# Download NLTK resources if necessary

try:import pandas as pd

import re

import spacy

import nltk

from nltk.*corpus* import stopwords # Import stopwords here

# Download NLTK resources if necessary

try:

stopwords.*words*('english')

except LookupError:

nltk.*download*('stopwords')

# Load the spaCy English model

try:

nlp = spacy.*load*("en\_core\_web\_sm")

except OSError:

import os

os.*system*("python -m spacy download en\_core\_web\_sm")

nlp = spacy.*load*("en\_core\_web\_sm")

**4.2.2) Loading the Dataset :**

* The CSV file is loaded using pandas. If there is an issue, an error message is displayed, and the program exits.
* The column product\_id is set to the value of the Product Name for identification purposes.

# Load the dataset

file\_path = r'D:/Bala DS/Data science Class materials/Projects/P6\_Final\_Project/Project\_Final/Dataset/Mobile\_Phone\_Data.csv'

try:

df = pd.*read\_csv*(file\_path)

print(f"Loaded dataset with {df.*shape*[0]} reviews.")

except Exception as e:

print(f"Error loading the dataset: {e}")

exit()

# Assign 'product\_id' as Product Name

df['product\_id'] = df['Product Name']

**4.2.3) Data Cleaning :**

* Rows with missing values in important columns like Review, Rating, and Price are removed.
* A function clean\_review is defined to clean the Review text by removing HTML tags, emojis, special characters, and converting the text to lowercase.

# Remove rows with missing values in 'Review', 'Rating', and 'Price'

df.*dropna*(subset=['Review', 'Rating', 'Price'], inplace=True)

# Cleaning the Review Text

def clean\_review(text):

"""Remove HTML tags, special characters, emojis, and convert to lowercase."""

# Remove HTML tags

text = re.*sub*(r'<.\*?>', '', text)

# Remove emojis using regex

text = re.*sub*(r'[^\x00-\x7F]+', '', text) # Removes non-ASCII characters (including emojis)

# Remove special characters, numbers (only keep alphabets)

text = re.*sub*(r'[^a-zA-Z\s]', '', text)

# Convert to lowercase

text = text.*lower*()

# Remove extra spaces

text = re.*sub*(r'\s+', ' ', text).*strip*()

return text

# Apply cleaning function to reviews

df['Cleaned\_Review'] = df['Review'].*apply*(clean\_review)

**4.2.4) Tokenization, Stopword Removal, and Lemmatization :**

* Using the spaCy NLP model, the review text is tokenized, stopwords are removed, and the remaining tokens are lemmatized (reduced to their base form).
* The tokenize\_and\_normalize function implements this process.

# Tokenization, stopword removal, and lemmatization

stop\_words = set(stopwords.*words*('english'))

def tokenize\_and\_normalize(text):

"""Tokenize, remove stopwords, and apply lemmatization."""

doc = nlp(text) # Tokenize and lemmatize using spaCy

# Remove stopwords, punctuation, and retain only alphabetic tokens

words = [token.*lemma\_* for token in doc if not token.*is\_stop* and token.*is\_alpha*]

return ' '.*join*(words)

# Apply tokenization and normalization

df['Tokenized\_Review'] = df['Cleaned\_Review'].*apply*(tokenize\_and\_normalize)

**4.2.5) Handling Numeric Fields:**

* The Rating column is converted to numeric format, and rows with invalid ratings are removed.
* The Price column is cleaned by removing non-numeric characters (like currency symbols) and converted to a numeric data type.
* The dataset is filtered to include only products priced between ₹20,000 and ₹40,000.

# Convert 'Rating' to numeric and filter out invalid entries

df['Rating'] = pd.*to\_numeric*(df['Rating'], errors='coerce')

df = df[df['Rating'].*notnull*()]

# Convert 'Price' to numeric (remove currency symbols and commas)

df['Price'] = df['Price'].*apply*(lambda x: re.*sub*(r'[^\d]', '', str(x))) # Remove non-numeric characters

df['Price'] = pd.*to\_numeric*(df['Price'], errors='coerce') # Convert to numeric

# Filter rows where the price is between ₹20,000 and ₹40,000

df = df[(df['Price'] >= 20000) & (df['Price'] <= 40000)]

**4.2.6) Final Steps :**

* Duplicates are removed if necessary, and the Product Link column is dropped as it is not needed for further analysis.
* The cleaned dataset is saved to a new CSV file.

# Drop duplicates if necessary

df.*drop\_duplicates*(inplace=True)

# Remove the 'Product Link' column

df.*drop*(columns=['Product Link'], inplace=True)

# Save the cleaned and processed data to a new CSV file

output\_path = 'D:/Bala DS/Data science Class materials/Projects/P6\_Final\_Project/Project\_Final/Processed\_Mobile\_Phone\_Data.csv'

df.*to\_csv*(output\_path, index=False)

print(f"Processed data saved to {output\_path}.")

# Display the first few rows of the cleaned data

print(df.*head*())

**5) Sentimental Analysis :**

**5.1) Libraries Used:**

* **pandas**: To load and manipulate the dataset.
* **TextBlob**: To perform sentiment analysis on the review text.
* **matplotlib** & **seaborn**: For visualizing sentiment distributions

**5.2) Steps for Sentimental Analysis**

**5.2.1) Loading the Preprocessed Data**

The preprocessed dataset (Processed\_Mobile\_Phone\_Data01.csv) is loaded using pandas for further analysis.

import pandas as pd

from textblob import TextBlob

import matplotlib.*pyplot* as plt

import seaborn as sns

# Load the preprocessed dataset

file\_path = 'D:/Bala DS/Data science Class materials/Projects/P6\_Final\_Project/Project\_Final/Processed\_Mobile\_Phone\_Data01.csv'

df = pd.*read\_csv*(file\_path)

**5.2.2) TextBlob Sentiment Analysis :**

* A function get\_textblob\_sentiment is defined to calculate the polarity score of the review text using TextBlob. The polarity score ranges from -1 (most negative) to 1 (most positive).
* The function also categorizes the review into five sentiment categories: *Strongly Positive*, *Positive*, *Neutral*, *Negative*, and *Strongly Negative* based on the polarity score.

# TextBlob Sentiment Analysis Function

def get\_textblob\_sentiment(text):

"""Determine sentiment polarity using TextBlob."""

analysis = TextBlob(text)

polarity = analysis.*sentiment*.*polarity* # Polarity score (-1 to 1)

# Categorize based on polarity thresholds

if polarity > 0.5:

sentiment = 'Strongly Positive'

elif 0 < polarity <= 0.5:

sentiment = 'Positive'

elif polarity == 0:

sentiment = 'Neutral'

elif -0.5 <= polarity < 0:

sentiment = 'Negative'

else:

sentiment = 'Strongly Negative'

return polarity, sentiment

**5.2.3) Applying Sentiment Analysis**

* The get\_textblob\_sentiment function is applied to the Cleaned\_Review column, generating a sentiment polarity score and corresponding sentiment category for each review.
* The results are saved into two new columns: TextBlob\_Sentiment\_Score and TextBlob\_Sentiment.

# Apply TextBlob sentiment analysis

df['TextBlob\_Sentiment\_Score'], df['TextBlob\_Sentiment'] = zip(\*df['Cleaned\_Review'].*apply*(get\_textblob\_sentiment))

# Save the updated DataFrame with TextBlob sentiment analysis

output\_path = 'D:/Bala DS/Data science Class materials/Projects/P6\_Final\_Project/Project\_Final/FlipKart\_Dataset\_with\_Sentiment\_Analysis.csv'

df.*to\_csv*(output\_path, index=False)

print(f"Sentiment analysis results with TextBlob saved to {output\_path}.")

**5.2.4) Visualization :**

* Sentiment Distribution: A count plot is created using seaborn to visualize the distribution of different sentiment categories across the dataset.
* Sentiment Score Histogram: A histogram of the polarity scores (ranging from -1 to 1) is generated to show the sentiment score distribution.

# Visualization: TextBlob Sentiment Distribution

plt.*figure*(figsize=(10, 5))

sns.*countplot*(x='TextBlob\_Sentiment', data=df, palette='pastel',

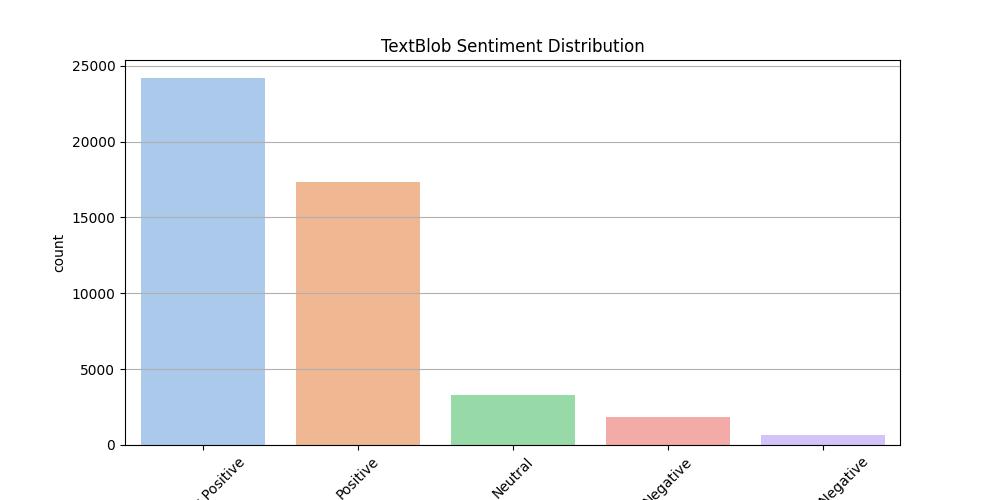
order=['Strongly Positive', 'Positive', 'Neutral', 'Negative', 'Strongly Negative'])

plt.*title*('TextBlob Sentiment Distribution')

plt.*xticks*(rotation=45)

plt.*grid*(axis='y')

plt.*show*()



# Histogram of TextBlob Sentiment Scores

plt.*figure*(figsize=(10, 5))

sns.*histplot*(df['TextBlob\_Sentiment\_Score'], bins=30, kde=True, color='blue')

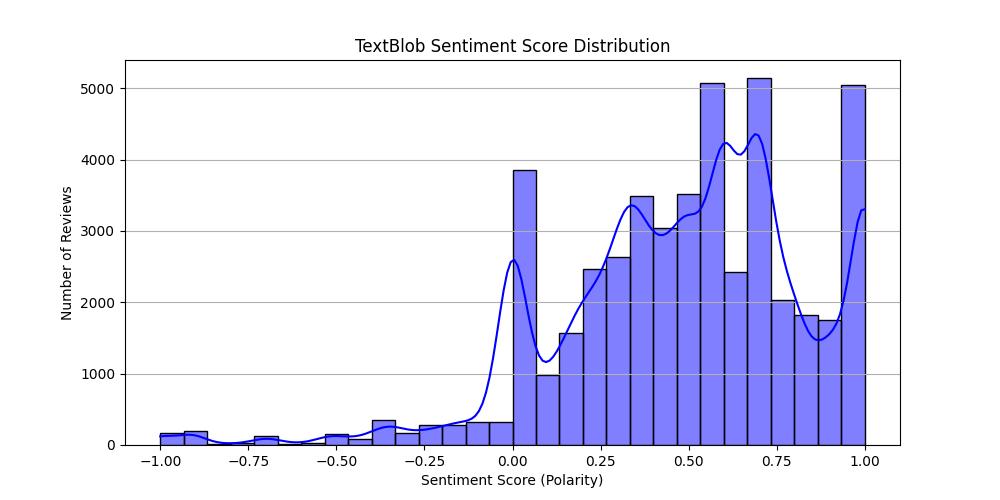
plt.*title*('TextBlob Sentiment Score Distribution')

plt.*xlabel*('Sentiment Score (Polarity)')

plt.*ylabel*('Number of Reviews')

plt.*grid*(axis='y')

plt.*show*()



**5.2.5) Product-Level Sentiment Summary:**

* The sentiment scores are aggregated at the product level to calculate the average sentiment score for each product and the total number of positive reviews.
* The top 5 products based on the highest average sentiment score are displayed.

# Aggregation: Average Sentiment Score by Product

product\_sentiment\_summary = df.*groupby*('Product Name').*agg*(

avg\_textblob\_sentiment\_score=('TextBlob\_Sentiment\_Score', 'mean'),

total\_textblob\_positive\_reviews=('TextBlob\_Sentiment', lambda x: (x == 'Positive').sum()),

total\_reviews=('TextBlob\_Sentiment', 'count') # Using TextBlob\_Sentiment for review count

).*sort\_values*(by='avg\_textblob\_sentiment\_score', ascending=False)

# Show the top 5 products based on TextBlob average sentiment score

print("Top 5 Products Based on TextBlob Sentiment Analysis:")

print(product\_sentiment\_summary.*head*())

**5.2.6) Saving Sentiment Summary:**

The sentiment summary, including the average sentiment score and positive review count per product, is saved as a new CSV file for future use, such as in a recommendation system.

# Save the product sentiment summary for future use in recommendation systems

summary\_output\_path = 'D:/Bala DS/Data science Class materials/Projects/P6\_Final\_Project/Project\_Final/Product\_Sentiment\_Summary\_TextBlob.csv'

product\_sentiment\_summary.*to\_csv*(summary\_output\_path)

print(f"Product sentiment summary saved to {summary\_output\_path}.")

**6) Data Visualization:**

**6.1) Libraries Used:**

* **matplotlib**: To create static visualizations and plots.
* **seaborn**: To generate aesthetically pleasing statistical plots built on top of matplotlib.

**6.2) Steps for Visualization:**

**6.2.1) Rating Distribution:**

A bar plot is created to show the distribution of product ratings using sns.countplot. It gives insights into the number of reviews for each rating level.

# Plot Rating Distribution

plt.*figure*(figsize=(10, 6))

sns.*countplot*(x='Rating', data=df, palette='viridis')

plt.*title*('Distribution of Ratings')

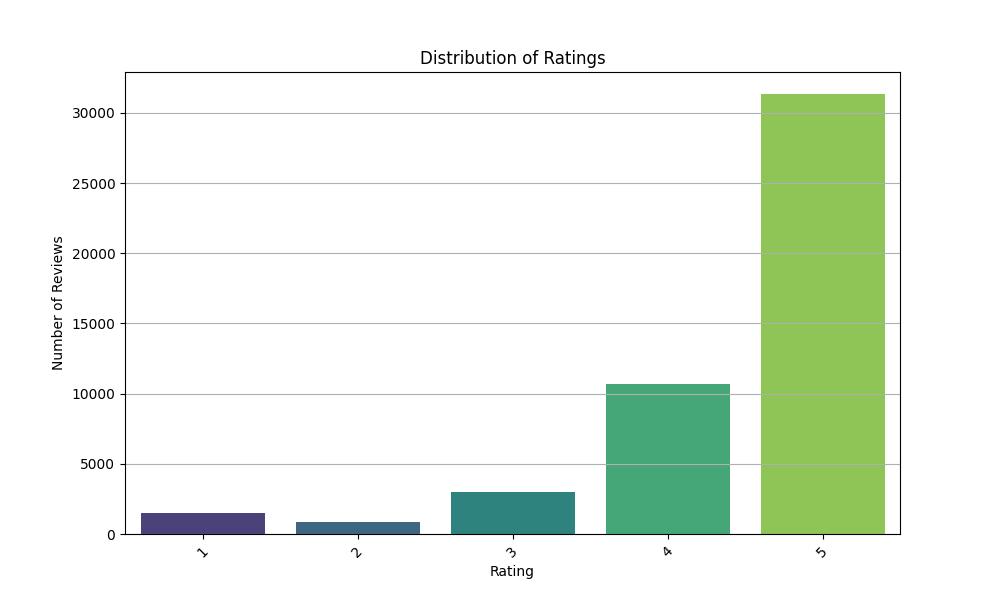
plt.*xlabel*('Rating')

plt.*ylabel*('Number of Reviews')

plt.*xticks*(rotation=45) # Rotate x-axis labels for better readability

plt.*grid*(axis='y') # Add a grid along the y-axis

plt.*show*()



**Interpretation**: This visualization helps understand how ratings are distributed (e.g., how many 5-star reviews, 4-star reviews, etc.) across products.

**6.2.2) Price Distribution:**

A histogram with a KDE (Kernel Density Estimate) plot is generated using sns.histplot to visualize the distribution of product prices.

# Plot Price Distribution

plt.*figure*(figsize=(10, 6))

sns.*histplot*(df['Price'], bins=30, kde=True, color='skyblue')

plt.*title*('Price Distribution of Products')

plt.*xlabel*('Price (₹)')

plt.*ylabel*('Number of Products')

plt.*grid*(axis='y')

plt.*show*()



**Interpretation**: This plot shows the distribution of product prices and helps identify pricing patterns, such as most products being in a particular price range.

**6.2.3) Top 10 Products by Average Rating:**

The average rating for each product is calculated using groupby and agg functions. The top 10 products by average rating are then visualized in a horizontal bar plot.

# Calculate average rating for each product

avg\_rating = df.*groupby*('product\_id').*agg*(Avg\_Rating=('Rating', 'mean')).*reset\_index*()

# Get the top 10 products by average rating

top\_n = avg\_rating.*sort\_values*(by='Avg\_Rating', ascending=False).*head*(10)

# Horizontal Bar Plot for Top N Products

plt.*figure*(figsize=(10, 6))

sns.*barplot*(x='Avg\_Rating', y='product\_id', data=top\_n, palette='coolwarm')

plt.*title*('Top 10 Products by Average Rating')

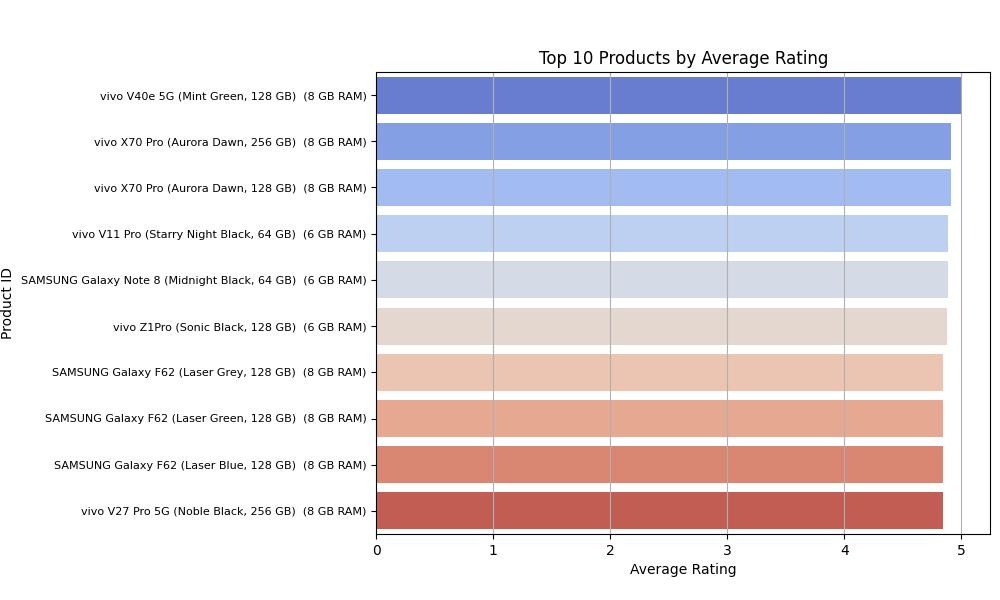
plt.*xlabel*('Average Rating')

plt.*ylabel*('Product ID')

plt.*yticks*(fontsize=8) # Adjust the font size for y-axis labels

plt.*grid*(axis='x')

plt.*show*()



**Interpretation**: This plot highlights the top-rated products based on average customer reviews, which is crucial for identifying the most well-received products.

**7) LangChain & Streamlit Implementation**

**7.1) Libraries and Setup**

* **streamlit**: To create an interactive web application.
* **pandas**: For data manipulation, loading, and filtering the sentiment data.
* **cohere**: For generating product recommendations based on natural language input from the user.
* **langchain\_community.llms**: Enables connecting to Cohere LLM via LangChain.

import streamlit as st

import pandas as pd

from langchain\_community.*llms* import Cohere

import cohere

import os

**7.2) Loading Sentiment Data:**

Sentiment Data: The app loads preprocessed review sentiment data, including Product Name, Price, and TextBlob\_Sentiment\_Score.

# Load sentiment data

sentiment\_data = pd.*read\_csv*("D:/Bala DS/Data science Class materials/Projects/P6\_Final\_Project/Project\_Final/FlipKart\_Dataset\_with\_Sentiment\_Analysis.csv")

**7.3) Initializing Cohere API**

**Cohere**: It is used to generate keywords or search terms from the user's input, which are used to filter the sentiment data for relevant products.

# Initialize Cohere (Replace with your Cohere API Key)

cohere\_api\_key = 'LGqaNW3Tfcdb3A1rhmd3NOZI5jMsbpLJlnjS4ZsW'

co = cohere.*Client*(cohere\_api\_key)

**7.4) Product Recommendation Logic:**

* **Cohere Input Processing**: The user's query is sent to Cohere’s generate() method, which returns relevant keywords.
* **Filtering Products**: Products whose reviews match the keywords are selected, and those with the highest sentiment scores are recommended.
* **Error Handling**: Any issues during recommendation generation will be handled gracefully with error messages.

# Function to recommend products based on user query

def recommend\_products(query, sentiment\_data):

try:

# Analyze the user's input using Cohere

response = co.*generate*(prompt=query, model='command-xlarge-nightly')

# Process response (you can also use a custom NLP method if needed)

search\_keywords = response.*generations*[0].*text*.*strip*().*split*()

# Filter sentiment data based on query analysis (e.g., find related product names or positive reviews)

recommendations = sentiment\_data[sentiment\_data['Cleaned\_Review'].str.*contains*('|'.*join*(search\_keywords), case=False)]

# Sort products by sentiment score

recommendations = recommendations.*sort\_values*(by='TextBlob\_Sentiment\_Score', ascending=False)

# Return top 5 products (Name and Price only)

return recommendations[['Product Name', 'Price']].*head*(5)

except Exception as e:

st.*error*(f"An error occurred while processing your request: {e}")

return pd.*DataFrame*()

**7.5) Streamlit UI Setup:**

**7.5.1) Page Config and Header:**

* **set\_page\_config**: Sets the title and layout of the app.
* **Header and Subtitle**: These are HTML-styled markdown elements for better aesthetics.

st.*set\_page\_config*(page\_title="Product Recommendation", layout="centered")

st.*markdown*("<h1 style='text-align: center; color: #800080;'>📱Flipkart Live Data Product Recommendation System📱</h1>", unsafe\_allow\_html=True)

st.*markdown*("<h3 style='text-align: center; color: #808080;'>Find the top-rated phones as per user reviews and sentiment analysis!</h3>", unsafe\_allow\_html=True)

**7.5.2) User Input Section**

* Text Area: Allows users to describe their ideal phone features. This input is passed to the recommendation function.
* Button: Triggers the product recommendation when clicked.

# User input section with styled markdown

st.*markdown*("<h4 style='color: #800000;'>📝To receive the top phone recommendations, please input your preferences:📝</h4>", unsafe\_allow\_html=True)

st.*markdown*("<p style='color: #FF4500;'>For example: <b>best camera phone</b>, <b>lightweight</b>, <b>long battery life</b>, etc.</p>", unsafe\_allow\_html=True)

# Create a container for input elements

with st.*container*():

query = st.*text\_area*("Describe your ideal phone features or type:", height=100)

recommend\_button = st.*button*("🔍 Recommend")

**7.5.3) Recommendation Display**

* **Spinner**: A spinner is displayed while the app processes the recommendation.
* **Product Display**: Recommended products are displayed with their names and prices inside styled boxes.
* **Error Handling**: If no products match, a warning is shown.

if recommend\_button:

if len(query.*strip*()) == 0:

st.*warning*("Please enter a valid query.")

else:

with st.*spinner*("Fetching recommendations..."):

recommended\_products = recommend\_products(query, sentiment\_data)

if not recommended\_products.*empty*:

st.*markdown*("<h3 style='color: #32CD32;'>🎊🎁Here are the top 5 phones that we recommend:🎊🎁</h3>", unsafe\_allow\_html=True)

for idx, row in recommended\_products.*iterrows*():

st.*markdown*(f"<div style='border: 1px solid #00BFFF; border-radius: 10px; padding: 10px; margin: 10px 0; background-color: #1E1E1E;'><p style='font-size:18px; color:#FFFFFF;'><b>{row['Product Name']}</b> - <span style='color:#FF4500;'>₹{row['Price']}</span></p></div>", unsafe\_allow\_html=True)

else:

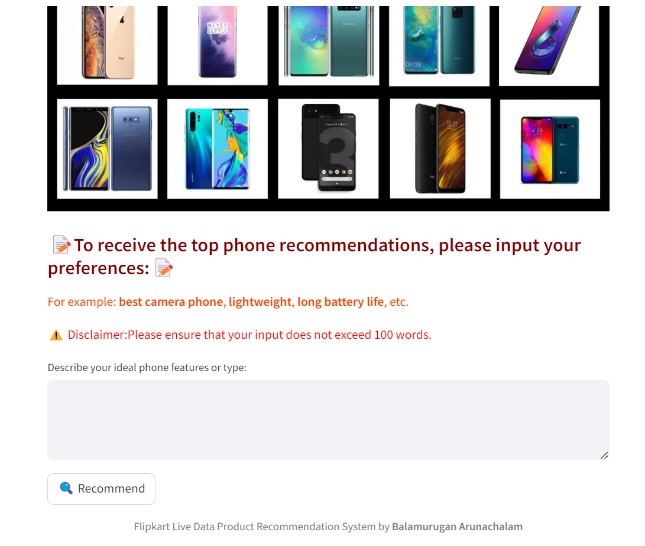
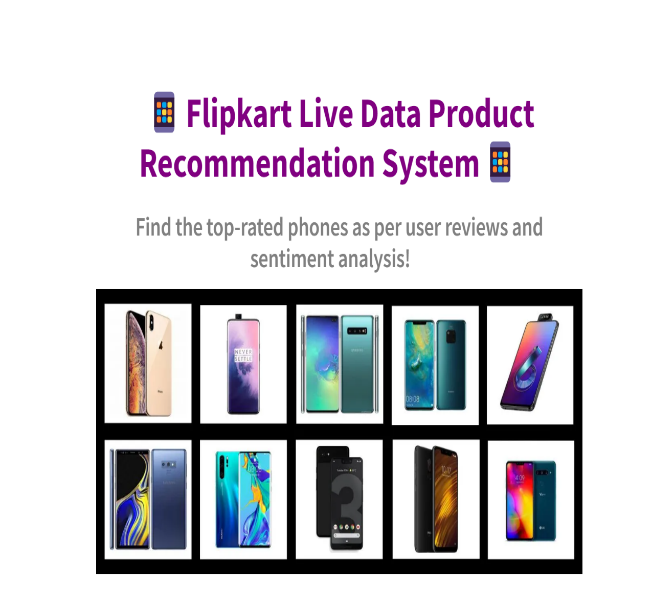
st.*warning*("No matching products found. Please try a different query.")

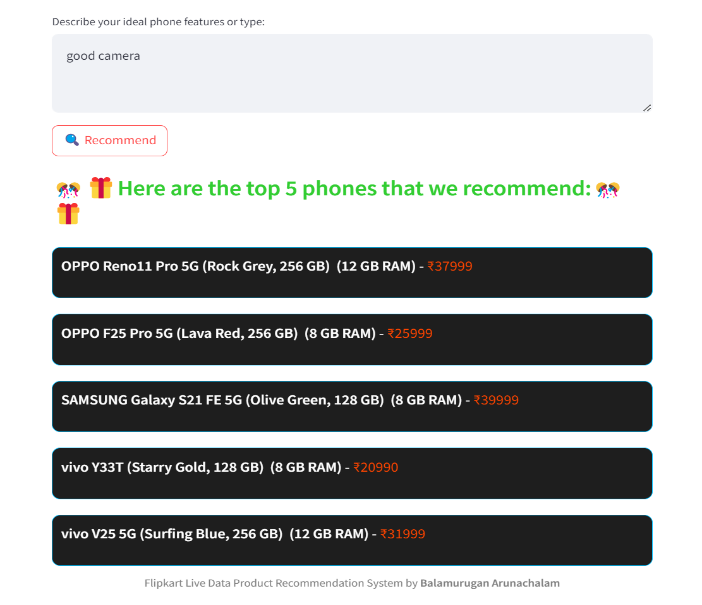
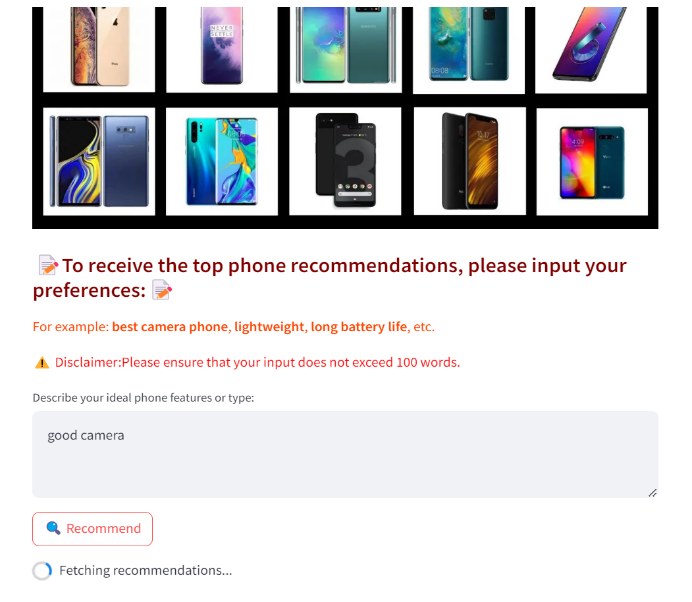
**7.5.4) Footer**

Displays the creator’s name at the bottom of the app.

st.*markdown*("<p style='text-align: center; font-size:14px; color:gray;'>Flipkart Live Data Product Recommendation System by <b>Balamurugan Arunachalam</b></p>", unsafe\_allow\_html=True)

**7.5.5) Screenshots:**





**8) Conclusion :**

The Web Scraping: Product Recommendation with Sentiment Analysis based on Reviews from Flipkart project successfully demonstrated the integration of various data science techniques to create an efficient recommendation system for e-commerce products. By leveraging web scraping techniques, we gathered real-time product reviews and ratings from Flipkart, enabling us to analyze user sentiments and preferences in a dynamic market environment.

**Key achievements of the project include:**

1. **Comprehensive Data Collection:** Utilizing Selenium and Beautiful Soup, we effectively scraped product details, reviews, and ratings, compiling a rich dataset that captures user opinions and product performance.
2. **Sentiment Analysis**: By employing natural language processing (NLP) techniques, we analyzed the sentiment of user reviews using TextBlob. This analysis provided valuable insights into customer satisfaction and product quality, allowing us to quantify sentiment scores associated with each product.
3. **Product Recommendation System:** The integration of LangChain facilitated the development of a product recommendation engine, which analyzes user queries and suggests top-rated products based on sentiment analysis. This feature enhances the user experience by providing tailored recommendations that align with customer preferences.
4. **User-Friendly Interface:** The Streamlit application provided an intuitive platform for users to input their preferences and receive personalized recommendations. The visually appealing interface and structured presentation of results contributed to an engaging user experience.
5. **Scalability and Deployment:** The project is designed to be scalable, with the potential for deployment on AWS, enabling real-time access to product data and recommendations. This scalability ensures the application can accommodate an expanding product range and user base.

In conclusion, this project not only highlights the capabilities of web scraping and sentiment analysis but also showcases the potential of leveraging these technologies to enhance e-commerce experiences. By providing consumers with informed product recommendations based on real user feedback, we empower them to make better purchasing decisions. Future enhancements could involve incorporating machine learning algorithms for improved recommendation accuracy and expanding the dataset to include additional e-commerce platforms.