SMART ELECTRONICS RECOMMENDATION SYSTEM

(COURSE CODE: 23UPCSC4P02)

Project With Viva Voce record submitted to Periyar University, Salem.
In partial fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE
IN
DATA SCIENCE

BY

MADESH M
REG NO: U23PG507DTS018

DEPARTMENT OF COMPUTER SCIENCE
PERIYAR UNIVERSITY

NAAC 'A++' Grade with CGPA 3.61 (Cycle - 3)

State University - NIRF Rank 59 - NIRF Innovation Band of 11-50

Salem-636011, Tamilnadu, India.

April – 2024.

Ms. R. SAMYA,
Department of Computer Science,
Periyar University,
Salem-11.

CERTIFICATE

This is to certify that the report of **"Project With Viva Voce" (23UPCSC4P02)** entitled "**SMART ELECTRONICS RECOMMENDATION SYSTEM"** submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science to the Periyar University, Salem is a record of bonafide work carried out by **MADESH M,** Register No. **U23PG507DTS018** under my supervision and guidance.

Signature of the Guide Signature of the HOD

Submitted of the Viva-Voce Examination held on _____.

Internal Examiner External Examiner

DECLARATION

I hereby, declare that the project work entitled " **Smart Electronics Recommendation System**" submitted to Periyar University in partial fulfilment of the requirement for the award of the Degree of **Master of Science in Data Science** is the record work carried out by me, under the supervision **MS. R. Samya, Periyar University**. To the best of my knowledge, the work reported here is not a part of any other work on the basis of which a degree or award was conferred on an earlier to one or any other candidate.

Place: Salem **Signature of Student**

Date:

ACKNOWLEDGMENT

First, I would like to thank the almighty for providing mw with everything that I requested to completing the project.

I would like to extend my sincere thanks to **Dr. R. JAGANNATHAN**, Vice Chancellor Periyar University, Salem who has been an invaluable source of providing the facility to complete the project successfully.

I would like to extend my sincere thanks to **Dr. C. CHANDRASEKAR**, Head of the Department, Department of Computer Science, Periyar University, Salem, For the support and encourage.

I would also like to express my special thanks to **DR. I. LAURANCE AROQUIARAJ**, Professor, Department of Computer Science, Periyar University, for his valuable guidance and encouragement throughout my academic journey.

I express my sincere thanks to my guide **MS. R. SAMYA**, department of Computer Science, Periyar University, Salem for the motivation and kind suggestion given in every step throughout this dissertation work and for valuable support in finishing this dissertation.

I extend my thanks to my parents and friends for their constant support and encouragement.

Place: Salem **Signature of the Student**

Date:

ABSTRACT

The **Smart Product Recommendation System** is an intelligent, AI-driven solution designed to enhance the online shopping experience by providing personalized product recommendations. This system leverages **Natural Language Processing (NLP)** techniques, specifically **TF-IDF vectorization and cosine similarity**, to analyse product titles, categories, brands, and features.

The recommendation engine identifies the most relevant products based on user queries, taking into account factors like **price range, brand, category, and user ratings**. Additionally, the system provides accessory recommendations, ensuring that users receive complementary product suggestions tailored to their needs.

Developed with **Streamlit** for an interactive user interface, the system integrates **fuzzy string matching** (via the FuzzyWuzzy library) to refine product searches and improve the accuracy of recommendations. By implementing **data preprocessing and feature engineering**, the model efficiently handles missing values and standardizes textual data.

This project demonstrates the power of **machine learning and NLP** in e-commerce, offering a scalable and adaptable solution for businesses looking to optimize customer engagement and improve product discovery.

Table of Contents

CHAPTER - 1

INTRODUCTION

1.1 Overview

In today's digital era, recommendation systems play a crucial role in helping users navigate vast amounts of data efficiently. Whether in e-commerce, streaming services, or online learning platforms, personalized recommendations enhance user experience by providing relevant content based on preferences and behaviour. This project aims to develop a **Recommendation System** that intelligently suggests items to users, improving engagement and satisfaction.

1.2 Objective

The primary goal of this project is to design and implement an intelligent recommendation system that predicts user preferences and provides personalized suggestions. The system will leverage **machine learning techniques** such as **Collaborative Filtering, Content-Based Filtering, and Hybrid Models** to generate accurate recommendations.

1.3 Motivation

With the increasing volume of online content, users often struggle to find relevant products, movies, music, or books. This recommendation system aims to:

- Enhance user experience by offering personalized suggestions.
- Reduce search time by providing relevant recommendations.
- Improve engagement and conversion rates in business applications.

1.4 Scope

The project will focus on:

✅ Collecting and processing user interaction data.

✅ Implementing **Collaborative Filtering, Content-Based Filtering, and Hybrid Models**.

✅ Evaluating model performance using metrics like **Precision, Recall, RMSE, and F1 Score**.

✅ Developing a **user-friendly interface** to display recommendations.

1.5 Expected Outcome

- A functional recommendation system capable of providing personalized suggestions.
- Improved accuracy and relevance of recommendations using **machine learning algorithms**.
- A scalable architecture that can be applied to multiple domains such as **movies, e-commerce, and music streaming**.

1.6 Core Applications

Recommendation systems power diverse industries:

- **E-commerce**: Product recommendations (e.g., Amazon, eBay).
- **Media & Entertainment**: Content suggestions (e.g., Netflix, Spotify).
- **Social Media**: Friend/connection recommendations (e.g., LinkedIn, Facebook).
- **News & Publishing**: Personalized article feeds (e.g., Google News).

CHAPTER - 2

PROBLEM IDENTIFICATION

2.1 Objectives of Recommendation Systems:

The primary objective of this **Recommendation System** is to develop an intelligent and personalized suggestion mechanism that enhances user experience by predicting preferences based on past interactions and behaviours.

The system aims to:

- **Provide Personalized Recommendations** – Suggest relevant items to users based on their preferences, browsing history, and interactions.
- **Enhance User Engagement** – Increase user interaction and satisfaction by minimizing search efforts and delivering meaningful recommendations.
- Implement Machine Learning Models – Utilize techniques like Collaborative Filtering, Content-Based Filtering, and Hybrid Models to improve recommendation accuracy.
- **Optimize Performance Metrics** – Evaluate the system using **Precision, Recall, RMSE (Root Mean Squared Error), and F1 Score** to ensure high-quality recommendations.
- **Develop a Scalable and Flexible Architecture** – Design a system that can be applied across various domains such as **e-commerce, movie streaming, music platforms, and online learning**.

The ultimate goal is to create an efficient, data-driven recommendation system that improves decision-making and enhances user satisfaction across different platforms. 🚀

2.2 Problem statement:

With the rapid growth of digital platforms, users are often overwhelmed by the vast amount of available content, products, or services. Manually searching for relevant items can be time-consuming and inefficient. Businesses and platforms struggle to engage users and provide meaningful recommendations that align with their interests.

Challenges:

- Users face difficulty in discovering relevant content/products due to information overload.
- Traditional search mechanisms do not offer personalized experiences.
- Businesses struggle with user retention and engagement without an effective recommendation system.
- Providing accurate, real-time, and scalable recommendations requires advanced machine learning techniques.

Proposed Solution:

This project aims to develop a Recommendation System that leverages machine learning algorithms to provide personalized suggestions based on user preferences and behaviours. The main concept of Collaborative Filtering, Content-Based Filtering, and Hybrid Models are used to analyse the past interactions and also predict the future preferences. The key features to maintain the and develop the user in various factors such as find relevant content easily, improve user engagement and satisfaction, assist business product to increase conversions and retention rates and finally to optimize recommendation accuracy through ML-based techniques.

By implementing this recommendation system, the user aims to enhance user experience and decision making, making content discovery a gained the knowledge to gain product-based information in a business manner. Research Question:

2.3 Research Question & Its Overcoming Solution:

RQ1. How accurately does the recommendation system suggest relevant products based on user preferences and filters?

RQ2. What is the impact of incorporating filters on the quality and diversity of product in the recommendations?

RQ3. How does the recommendation system's performance vary with different similarity metrics or techniques?

CHAPTER - 3

Overall System Design

3.1 System configuration:

The **Recommendation System** requires a well-defined hardware and software environment for effective development, training, and deployment. Below are the necessary system specifications.

Operating System:

The project can be developed and deployed on multiple operating systems, including:

- ✅ **Windows 10/11**
- ✅ **Linux (Ubuntu 20.04 or later)**
- ✅ **macOS (10.15 or later)**

Programming Language:

✅ **Python** is the primary language for implementing the recommendation system due to its extensive ecosystem for data science, machine learning, and deep learning.

Machine Learning & Deep Learning Frameworks:

- ✅ **Scikit-learn** – For traditional machine learning models like Collaborative Filtering
- ✅ **TensorFlow / PyTorch** – For deep learning-based recommendation models
- ✅ **Surprise Library** – For matrix factorization and collaborative filtering algorithms

---

Libraries & Tools:

- ✅ **NumPy & Pandas** – For data manipulation and preprocessing
- ✅ **Scikit-learn** – For ML algorithms and evaluation metrics
- ✅ **Matplotlib & Seaborn** – For data visualization

✅ **OpenCV** – If image-based recommendations are required

✅ **NLTK or spaCy** – If text-based recommendations are needed

✅ **Flask / FastAPI / Django / Streamlit** – For building and deploying APIs

---

Development Environment:

The recommendation system can be developed using the following IDEs:

✅ **Jupyter Notebook** – For experimentation and testing

✅ **PyCharm** – For structured development

✅ **Visual Studio Code** – Lightweight and extensible IDE

3.2 MY SYSTEM SPECIFICATION:

3.2.1 SOFTWARE SPECIFICATION:

Operating System	Windows 11
Coding Language	Python Language
Software Tool	Anaconda, Jupyter Notebook.
Cloud Platforms	AWS, Google Colab

3.2.2 HARDWARE CONFIGURATION:

Processor	12th Gen Intel(R) Core(TM) i5-1240P 1.70 GHz
Installed RAM	16.0 GB
Hard Disk	512 GB

CHAPTER - 4

Description of Modules

4.1 RECOMMENDATION SYSTEM:

A recommendation system (or **recommender system**) is a type of software tool or algorithm designed to predict and suggest items (such as products, movies, books, or content) to users based on various factors. These factors typically include user preferences, behaviors, past interactions, and similarities between items or users.

4.1.1 TYPE OF RECOMMENDATION SYSTEM:

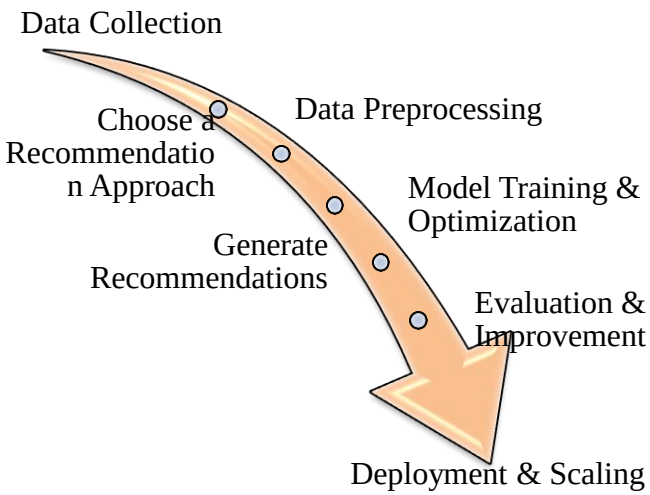
1. Collaborative Filtering: This method makes recommendations based on the past behavior of similar users. It assumes that if two users have similar tastes in the past, they will also prefer similar items in the future.

- **User-based**: Recommends items that similar users have liked.
- **Item-based**: Recommends items that are similar to items the user has liked.

2. Content-Based Filtering: This method recommends items similar to those the user has liked in the past, based on features or attributes of the items (e.g., genre, keywords, or product specifications).

3. Hybrid Methods: These systems combine collaborative filtering and content-based filtering to improve recommendations by leveraging the strengths of both approaches.

4.2 WORKING MECHANISM:



4.2.1 DATA COLLECTION

● The goal is to gather product data from Amazon for Apple and Samsung products.

● web scraping techniques are used to extract relevant product details from the Amazon website.

● This includes attributes such as Product Id, Product Name, Category, Price, Specifications, Reviews, Ratings and Availability

● Dataset containing all the necessary product attributes, will used for feature extraction and recommendation.

WEB SCRAPING:

Web scraping is the process of extracting data from websites using automated tools or scripts. It involves fetching a webpage, parsing its content, and extracting the desired information for analysis or storage.

PYTHON:

- **BeautifulSoup** : For parsing and extracting from HTML. Ø**Requests** : For sending HTTP requests.
- **Scrapy** : A powerful web scraping and crawling framework.
- **Selenium** : For scraping dynamic website that use JavaScript for content rendering.
- **Puppeteer** : A Node.js library for controlling headless browsers (e.g., Chore)

Objective:

Scraping product data for Apple and Samsung products to build a **content-based recommendation system**.

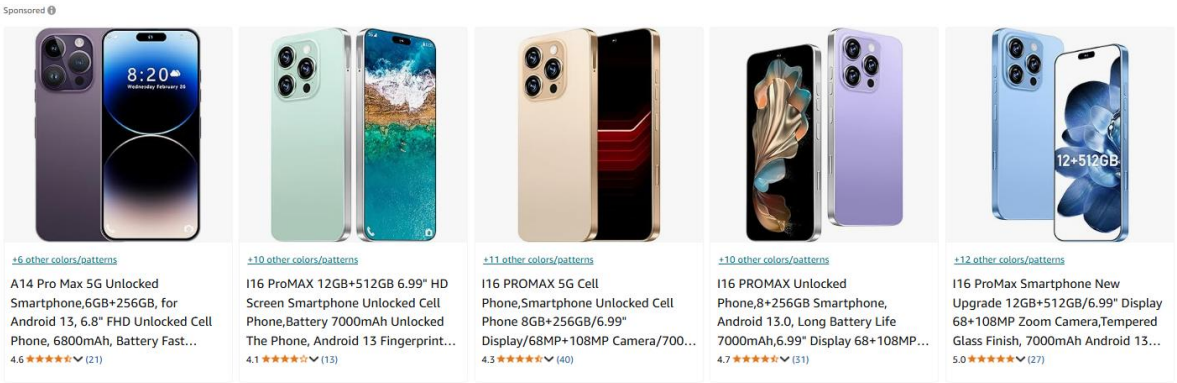
Example Websites:

Amazon (https://www.amazon.com/)

Products:

- Apple
- Samsung

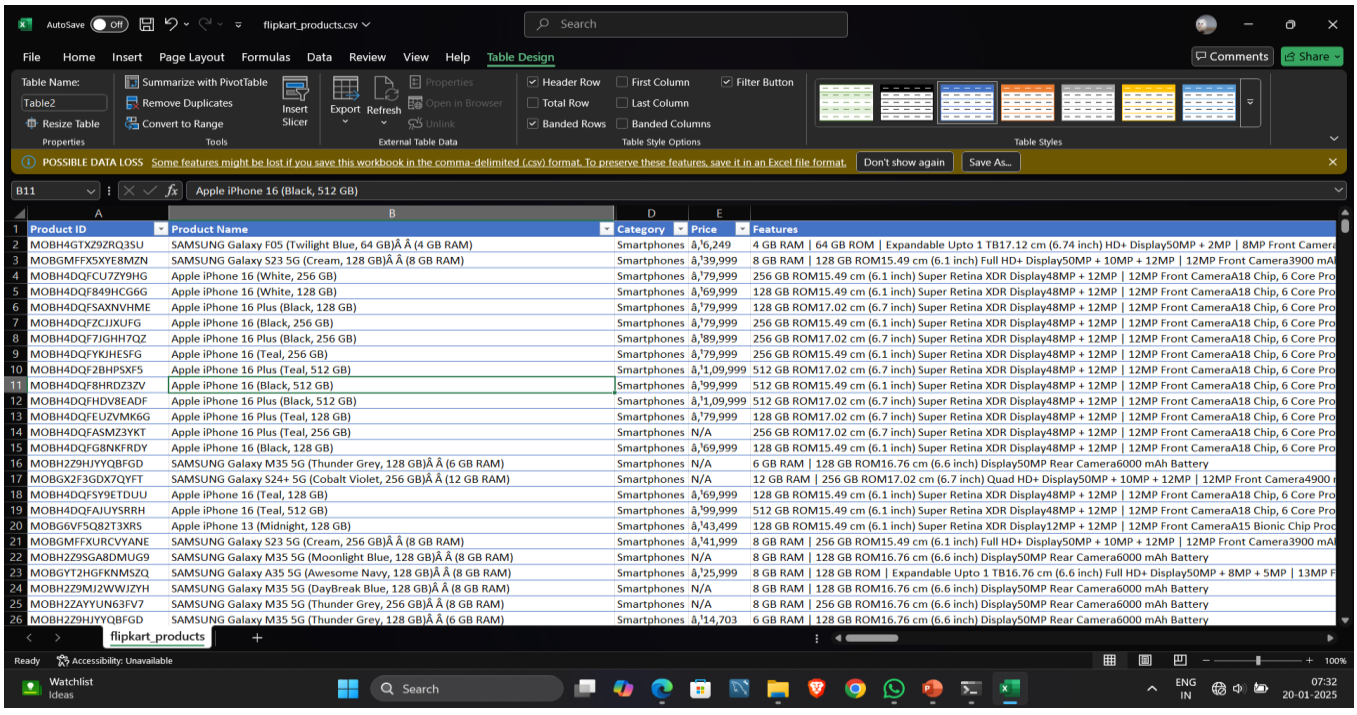
Dataset Link:https://github.com/madesh6554/Smart-Electronics-Recommendation-System/blob/main/Dataset_preprocessed_with_accessory.csv



Product Features:

Features	Details
Product id	Unique identifier for the product
Product name	Name of the product.
Brand	Manufacturer or brand name
Price	Cost of the product.
Description	Brief overview of the product.
category	Type or classification of the product
Specifications	Technical details or key features.
Features	Highlighted functionalities or benefits.
Ratings	Numerical or star-based score by users.

Sample Dataset:



Product ID	Product Name	Category	Price	Features
MOBH4GTXZ9ZRQ3SU	SAMSUNG Galaxy F05 (Twilight Blue, 64 GB)Â Â (4 GB RAM)	Smartphones	â‚¹6,249	4 GB RAM | 64 GB ROM | Expandable Upto 1 TB17.12 cm (6.74 inch) HD+ Display50MP + 2MP | 8MP Front Camera
MOBGMFFX5XYE8MZN	SAMSUNG Galaxy S23 5G (Cream, 128 GB)Â Â (8 GB RAM)	Smartphones	â‚¹39,999	8 GB RAM | 128 GB ROM15.49 cm (6.1 inch) Full HD+ Display50MP + 10MP + 12MP | 12MP Front Camera3900 mA
MOBH4DQFCU7ZY9HG	Apple iPhone 16 (White, 256 GB)	Smartphones	â‚¹79,999	256 GB ROM15.49 cm (6.1 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQF849HCG6G	Apple iPhone 16 (White, 128 GB)	Smartphones	â‚¹69,999	128 GB ROM15.49 cm (6.1 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQFSAXNVHME	Apple iPhone 16 Plus (Black, 128 GB)	Smartphones	â‚¹79,999	128 GB ROM17.02 cm (6.7 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQFZCJJXUFG	Apple iPhone 16 (Black, 256 GB)	Smartphones	â‚¹79,999	256 GB ROM15.49 cm (6.1 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQF7JGHH7QZ	Apple iPhone 16 Plus (Black, 256 GB)	Smartphones	â‚¹89,999	256 GB ROM17.02 cm (6.7 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQFYKJHESFG	Apple iPhone 16 (Teal, 256 GB)	Smartphones	â‚¹79,999	256 GB ROM15.49 cm (6.1 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQF2BHPSXF5	Apple iPhone 16 Plus (Teal, 512 GB)	Smartphones	â‚¹1,09,999	512 GB ROM17.02 cm (6.7 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQF8HRDZ3ZV	Apple iPhone 16 (Black, 512 GB)	Smartphones	â‚¹99,999	512 GB ROM15.49 cm (6.1 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQFHDV8EADF	Apple iPhone 16 Plus (Black, 512 GB)	Smartphones	â‚¹1,09,999	512 GB ROM17.02 cm (6.7 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQFEUZVMK6G	Apple iPhone 16 Plus (Teal, 128 GB)	Smartphones	â‚¹79,999	128 GB ROM17.02 cm (6.7 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQFASMZ3YKT	Apple iPhone 16 Plus (Teal, 256 GB)	Smartphones	N/A	256 GB ROM17.02 cm (6.7 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQFG8NKFRDY	Apple iPhone 16 (Black, 128 GB)	Smartphones	â‚¹69,999	128 GB ROM15.49 cm (6.1 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH2Z9HJYYQBFGD	SAMSUNG Galaxy M35 5G (Thunder Grey, 128 GB)Â Â (6 GB RAM)	Smartphones	N/A	6 GB RAM | 128 GB ROM16.76 cm (6.6 inch) Display50MP Rear Camera6000 mAh Battery
MOBGX2F3GDX7QYFT	SAMSUNG Galaxy S24+ 5G (Cobalt Violet, 256 GB)Â Â (12 GB RAM)	Smartphones	N/A	12 GB RAM | 256 GB ROM17.02 cm (6.7 inch) Quad HD+ Display50MP + 10MP + 12MP | 12MP Front Camera4900
MOBH4DQFSY9ETDUU	Apple iPhone 16 (Teal, 128 GB)	Smartphones	â‚¹69,999	128 GB ROM15.49 cm (6.1 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBH4DQFAJUYSRRH	Apple iPhone 16 (Teal, 512 GB)	Smartphones	â‚¹99,999	512 GB ROM15.49 cm (6.1 inch) Super Retina XDR Display48MP + 12MP | 12MP Front CameraA18 Chip, 6 Core Pro
MOBG6VF5Q82T3XRS	Apple iPhone 13 (Midnight, 128 GB)	Smartphones	â‚¹43,499	128 GB ROM15.49 cm (6.1 inch) Super Retina XDR Display12MP + 12MP | 12MP Front CameraA15 Bionic Chip Proc
MOBGMFFXURCVYANE	SAMSUNG Galaxy S23 5G (Cream, 256 GB)Â Â (8 GB RAM)	Smartphones	â‚¹41,999	8 GB RAM | 256 GB ROM15.49 cm (6.1 inch) Full HD+ Display50MP + 10MP + 12MP | 12MP Front Camera3900 mA
MOBH2Z9SGA8DMUG9	SAMSUNG Galaxy M35 5G (Moonlight Blue, 128 GB)Â Â (8 GB RAM)	Smartphones	N/A	8 GB RAM | 128 GB ROM16.76 cm (6.6 inch) Display50MP Rear Camera6000 mAh Battery
MOBGYT2HGFKNMSZQ	SAMSUNG Galaxy A35 5G (Awesome Navy, 128 GB)Â Â (8 GB RAM)	Smartphones	â‚¹25,999	8 GB RAM | 128 GB ROM | Expandable Upto 1 TB16.76 cm (6.6 inch) Full HD+ Display50MP + 8MP + 5MP | 13MP F
MOBH2Z9MJ2WWJZYH	SAMSUNG Galaxy M35 5G (DayBreak Blue, 128 GB)Â Â (8 GB RAM)	Smartphones	N/A	8 GB RAM | 128 GB ROM16.76 cm (6.6 inch) Display50MP Rear Camera6000 mAh Battery
MOBH2ZAYYUN63FV7	SAMSUNG Galaxy M35 5G (Thunder Grey, 256 GB)Â Â (8 GB RAM)	Smartphones	N/A	8 GB RAM | 256 GB ROM16.76 cm (6.6 inch) Display50MP Rear Camera6000 mAh Battery
MOBH2Z9HJYYQBFGD	SAMSUNG Galaxy M35 5G (Thunder Grey, 128 GB)Â Â (6 GB RAM)	Smartphones	â‚¹14,703	6 GB RAM | 128 GB ROM16.76 cm (6.6 inch) Display50MP Rear Camera6000 mAh Battery

Scrape the data from website:

```
[4]: search_url = 'https://www.amazon.in/s?k=apple&rh=n%3A976419031%2Cp_123%3A110955&dc&ds=v1%3Ab6VLEEcmVLXZ%2BXSs5HgU1N8%2FmXBk7OlxJm3koCdyZw8&crid=2YSEGXA
base_url= 'https://www.amazon.in'

headers = {
    "User-Agent": 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36 Edg/132.0.0.0'
}


def get_product_links(url):

    apple_product_links = []
    r = requests.get(url, headers = headers)
    if r.status_code == 200:
        soup =BeautifulSoup(r.content, 'html.parser')
        for i in soup.select('a.a-link-normal.s-line-clamp-2.s-link-style.a-text-normal'):
            apple_product_links.append(base_url+i['href'])
    else:
        print(f"failed to fetch {url}")
    return apple_product_links


def get_next_page(soup):
    next_page = soup.select_one('a.s-pagination-item.s-pagination-next.s-pagination-button.s-pagination-button-accessibility.s-pagination-separator')
    if next_page and 'href' in next_page.attrs:
        return base_url+ next_page['href']
    return None

def scrape_amazon(search_url):
    all_product_links = []
    current_url = search_url
    while current_url:
        print(f'scaping: {current_url}')
        r = requests.get(current_url, headers = headers)
        if r.status_code == 200:
            soup = BeautifulSoup(r.content,'html.parser')
            all_product_links.extend(get_product_links(current_url))
            current_url = get_next_page(soup)
            time.sleep(2)
        else:
            print(f'Error fetching page:{r.status_code}')
            break
    return all_product_links
product_links = scrape_amazon(search_url)
print(len(product_links))
```

```
scaping: https://www.amazon.in/s?k=apple&rh=n%3A976419031%2Cp_123%3A110955&dc&ds=v1%3Ab6VLEEcmVLXZ%2BXSs5HgU1N8%2FmXBk7OlxJm3koCdyZw8&crid=2YSEGXAQ
YAID0&qid=1737545299&rnid=91049095031&sprefix=ap%2Caps%2C2464&ref=sr_nr_p_123_1
scaping: https://www.amazon.in/s?k=apple&i=electronics&rh=n%3A976419031%2Cp_123%3A110955&dc&page=2&xpid=JKwd6d1QjgZEt&crid=2YSEGXAQYAID0&qid=173754
6031&rnid=91049095031&sprefix=ap%2Caps%2C2464&ref=sr_pg_1
```

4.2.2 Preprocessing:

The goal is to Clean and preprocess the scraped data to focus on relevant and high-quality product information.

The Methods involved in filtering are:

- Removing duplicates
- Handling missing data
- Checking data consistency
- Filtering based on relevance

The filtered data will be ready for the next stage, where you can extract useful features and prepare for recommendations.

Scrape the product details using web scraping:

```
# Function to scrape product details
def scrape_product_details(url):
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        soup = BeautifulSoup(response.content, "html.parser")

        # Extract Title
        title = soup.select_one("#productTitle")
        product_title = title.text.strip() if title else "Title not found"

        base_title = re.sub(r"\(.*?\)", "", product_title).strip()
        base_title = re.sub(r"\s+-.*", "", base_title)
        # Extract Category
        # category = soup.select_one("ul.a-unordered-list.a-horizontal.a-size-small span.a-size-base")
        # product_category = category.text.strip() if category else "Category not found"
        product_category = 'Mobile Phones'

        # Extract Brand
        brand = soup.select_one("td.a-span9 span.a-size-base")
        product_brand = brand.text.strip() if brand else "Brand not found"

        # Extract Price
        price = soup.select_one(".a-price .a-offscreen")
        product_price = price.text.strip() if price else "Price not found"

        # Extract Released Year
        # release_date = soup.select_one("div#detailBullets_feature_div li:contains('Date First Available')")
        # released_year = release_date.text.strip()[-4:] if release_date else "Release year not found"
        release_date = "+".join(base_title.split())
        search_rurl = f"https://www.phonearena.com/search?term={release_date.replace(' ', '+')}"
        # Send a request to the search page
        response = requests.get(search_rurl)

       # Check if the request was successful
       if response.status_code == 200:
            soup = BeautifulSoup(response.content, "html.parser")

            # Find the anchor tag inside the 'tile-content' div
            link_tag = soup.select_one('.tile-content a')

            if link_tag:
                r_link = link_tag['href']
                product_rurl = r_link  # Complete the URL

                # Request the product page
                rl = requests.get(product_rurl)
                soup = BeautifulSoup(rl.text, 'html.parser')

                # Extract the released date
                released_date_tag = soup.select_one('.widgetQuickSpecs__title_paragraph')
                released_date = released_date_tag.text.strip() if released_date_tag else "Release date not found"

                print(f"Release year for {phone_name}: {released_date}")
            else:
                print("Link not found.")
       else:
            print(f"Failed to fetch search results for {phone_name}")
```

```
        # Extract Specifications
        specs = soup.select("#feature-bullets ul li")
        specifications = [spec.text.strip() for spec in specs if spec.text.strip()]

        # Timestamp
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        # Extract Availability
        availability = soup.select_one("#availability span")
        product_availability = availability.text.strip() if availability else "Availability not found"

        # Extract Product ID
        # product_id = url.split("/")[-1].split("?")[0]
        match = re.search(r'/dp/([A-Z0-9]{10})', url)
        if match:
            asin = match.group(1)

        # Check if Refurbished
        refurbished = "Yes" if "refurbished" in product_title.lower() else "No"

        # Return as a dictionary
        return {
            "Title": product_title,
            "Category": product_category,
            "Brand": product_brand,
            "Price": product_price,
            "Released Year": released_date,
            "Specifications": specifications,
            "Timestamp": timestamp,
            "Availability": product_availability,
            "Product ID": asin,
            "Refurbished": refurbished
        }
    else:
        print(f"Failed to fetch product details: {url}")
        return None
```

4.2.3 Model Techniques:

1. Content-Based Filtering:

Content-based filtering is a recommendation system technique that recommends items to users based on the attributes or features of the items themselves, rather than relying on the preferences of other users. This approach uses information about the item (such as its description, genre, keywords, etc.) to make personalized recommendations.

- The goal is to create a recommendation system that suggests products to users based on their features.
- **Feature Extraction** is used to extract relevant product attributes and process text data using TF-IDF.
- **Similarity Calculation** is performed using cosine similarity to identify similar products based on their features.

- **Content-Based Filtering** matches user preferences with similar product features to recommend relevant items.

Hence, the recommendation system provides personalized product recommendations by analysing feature similarities.

```
[18]: from sklearn.feature_extraction.text import TfidfVectorizer
      from sklearn.metrics.pairwise import cosine_similarity

[20]: vect = TfidfVectorizer(stop_words='english')
      tfidf_matix_1 = vect.fit_transform(df['Specifications'])
      similarity_matrix_1 = cosine_similarity(tfidf_matix_1, tfidf_matix_1)

[32]: def recommend_product(product_title, matrix, titles,pries, num_recommendations = 5):
          if product_title not in titles:
              return 'Product Not Found'
          idx = titles.index(product_title)
          similarity_score_1 = list(enumerate(matrix[idx]))
          similarity_score_1 = sorted(similarity_score_1, key = lambda x:x[1] , reverse=True)
          recommend_indies = [i[0] for i in similarity_score_1[1:num_recommendations+1]]
          return [(titles[i], pries[i]) for i in recommend_indies]

      product_titles = df['Title'].tolist()
      product_pries = df['Price'].tolist()
      ex = product_titles[122]
      recommend_products = recommend_product(ex, similarity_matrix_1, product_titles,product_pries)
      recommend_products

[32]: [('Apple iPhone 14 Plus (128 GB) - Starlight', '₹69,900.00'),
       ('Apple iPhone 14 Plus (128 GB) - Blue', '₹69,900.00'),
       ('Apple iPhone 14 Plus (128 GB) - (Product) RED', '₹69,900.00'),
       ('Apple iPhone 14 Plus (128 GB) - Purple', '₹69,900.00'),
       ('Apple iPhone 14 Plus (256 GB) - Blue', '₹72,900.00')]
```

2. Rule – Based Recommendation system:

A **Rule-Based Recommendation System** suggests items to users based on predefined rules rather than using machine learning models. It relies on explicit conditions set by domain experts or business logic.

- **Define Rules** – Set conditions based on user attributes, preferences, or behavior.
- **Match Rules to Items** – Use filtering (e.g., category-based, threshold-based).
- **Rank & Recommend** – Present top-matching items to users.

```
[14]: df['features'] = df['features'].apply(lambda x: ' '.join(x.replace('{', '').replace('}', '').replace("'", "").split(',')))

[15]: df[df['product_id']=='P03229']
```

[15]:

	product_id	product_name	product_category	brand	model	features	size	price	rating	review_count	compatibility	frequently_bought_together	ch
1653	P03229	MagSafe Clear Case	Case	Apple	Galaxy Z Flip5	material: Polycarbonate size_compatibility: 6...	6.7 inches	2499	4.8	2200	['Galaxy Z Flip5']	['Galaxy Z Flip5', '45W USB-C Adapter']	Nc

```
[17]: import ast

# Convert column from string representation of lists to actual lists
df['compatibility'] = df['compatibility'].apply(ast.literal_eval)
df['frequently_bought_together'] = df['frequently_bought_together'].apply(ast.literal_eval)

# Now safely convert lists to comma-separated strings
df['compatibility'] = df['compatibility'].apply(lambda x: ", ".join(x))
df['frequently_bought_together'] = df['frequently_bought_together'].apply(lambda x: ", ".join(x))

# print(df['compatibility'].head())
df
```

[17]:

	product_id	product_name	product_category	brand	model	features	size	price	rating	review_count	compatibility	frequently_bought_toge
0	P00001	25W Fast Charger	Charger	Apple	Universal	charging_type: USB-C wattage: 45W cable_leng...	Not Applicable	1900	4.7	4517	iPhone 13, Galaxy S23 Ultra, iPhone 14	iPhone 13, Galaxy S23 iPho
1	P00003	iPhone SE	Smartphone	Apple	iPhone SE	display_size: 6.7 inches processor: A15 Bioni...	6.7 inches	79999	4.8	4995	MagSafe Clear Case, S-View Flip Cover	MagSafe Clear Case, S-Flip (
2	P00004	Silicone Case	Case	Apple	Galaxy A54	material: PU Leather size_compatibility: 6.1 ...	6.1 inches	2999	4.7	425	Galaxy A54	Galaxy A54, 25W Fast Ch
3	P00005	45W USB-C Adapter	Charger	Apple	Universal	charging_type: USB-A wattage:	Not Applicable	2999	4.2	2960	Galaxy S23, iPhone SE, Galaxy S23	Galaxy S23, iPhone SE, G S23

```
[56]: def recommend_cases(user_phone, df):
          # Filter products where user_phone is in the compatibility list
          recommended = df[df['compatibility'].apply(lambda x: user_phone in x)]
          return recommended['product_name'].tolist()[1:5], recommended['product_id'].tolist()[1:5]

      # Example usage
      user_phone = '25W Fast Charger'
      recommendations = recommend_cases(user_phone, df)
      print(recommendations)
```

```
(['iPhone 13', 'iPhone 13', 'iPhone 13', 'iPhone SE'], ['P00045', 'P00051', 'P00058', 'P00069'])
```

3. Hybrid Model Recommendation system:

The hybrid recommendation model combines content-based filtering and rule-based filtering to enhance product recommendations. Content-based filtering uses TF-IDF vectorization and cosine similarity to identify similar products based on textual features such as product titles, categories, brands, and attributes. Rule-based filtering ensures that recommendations meet user-defined constraints such as price range, rating thresholds, brand preferences, and category selections. By integrating these approaches, the system provides personalized and context-aware suggestions while maintaining flexibility through user-defined rules. This combination improves recommendation accuracy by leveraging textual similarity and logical constraints, ensuring that recommended products are relevant, diverse, and aligned with user preferences.

```
# Function to get content-based recommendations
def content_based_recommendations(product_id, top_n=5):
    idx = df.index[df['product_id'] == product_id].tolist()[0]
    similarity_scores = list(enumerate(cosine_sim[idx]))
    similarity_scores = sorted(similarity_scores, key=lambda x: x[1], reverse=True)[1:top_n+1]
    recommended_indices = [i[0] for i in similarity_scores]
    return df.iloc[recommended_indices][['product_id', 'product_name']]
```

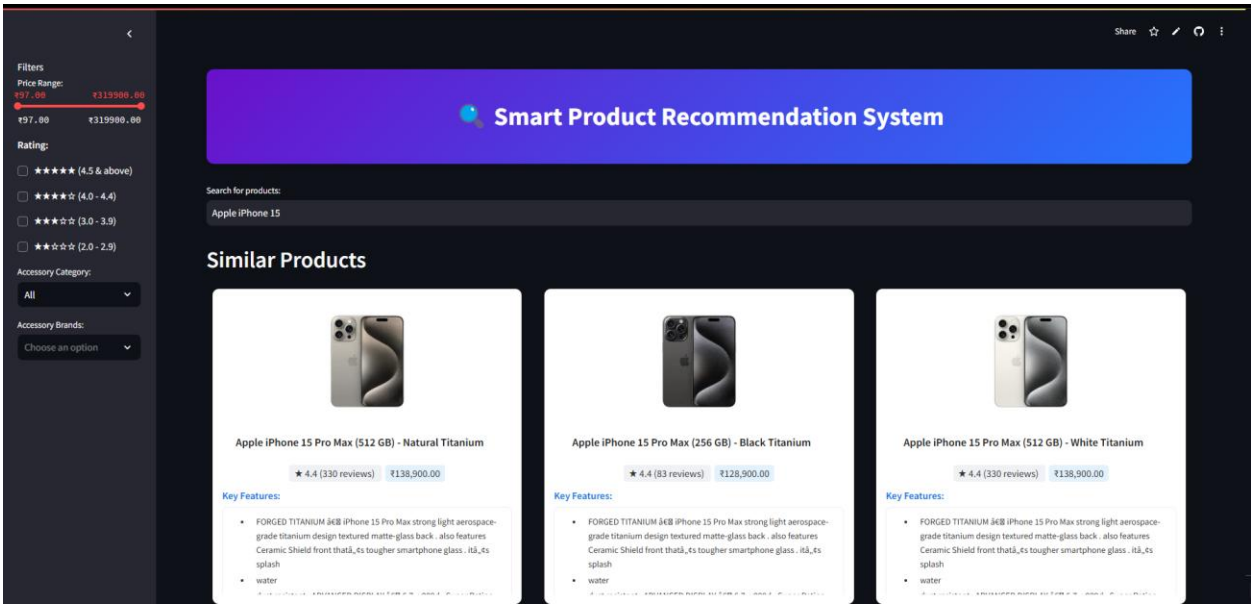
```python
# Function to get rule-based recommendations
def rule_based_recommendations(product_id):
    product_row = df[df['product_id'] == product_id].iloc[0]
    category = product_row['product_category']
    frequently_bought = product_row['frequently_bought_together']

    # Rule: Recommend frequently bought together products
    recommended_products = df[df['product_name'].isin(frequently_bought)][['product_id', 'product_name']]

    # Rule: If it's a smartphone, recommend cases and chargers
    if category == 'Smartphone':
        accessories = df[df['product_category'].isin(['Case', 'Charger'])][['product_id', 'product_name']]
        recommended_products = pd.concat([recommended_products, accessories]).drop_duplicates()

    return recommended_products.head(5)

# Hybrid recommendation function
def hybrid_recommendations(product_id, top_n=5):
    content_recs = content_based_recommendations(product_id, top_n)
    rule_recs = rule_based_recommendations(product_id)
    return pd.concat([content_recs, rule_recs]).drop_duplicates().head(top_n)

# Example usage
product_id = 'P00855'
recommendations = hybrid_recommendations(product_id, top_n=5)
print(recommendations)
```

```
     product_id product_name
1362     P02634    iPhone 14
2027     P04004    iPhone 14
2264     P04500    iPhone 14
2391     P04757    iPhone 14
2426     P04833    iPhone 14
```

4.2.4 USER INTERACTION

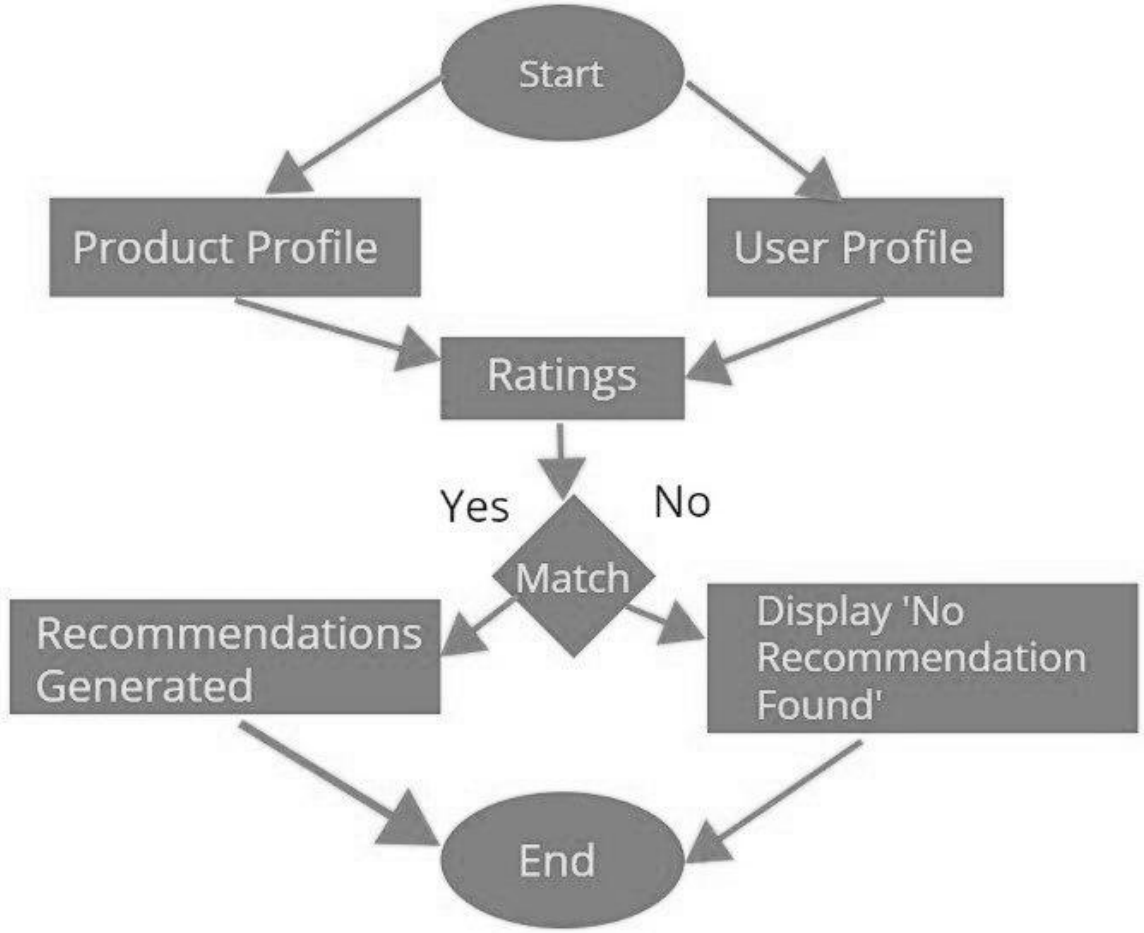
- The user specifies their preferences, such as brand, price range, specifications, or product type.
- Based on these inputs, the system processes the data and displays personalized product recommendations.

Project website link: https://smart-electronics-recommendation-system-cg6eyh2obhqeyju7vca6gn.streamlit.app/



CHAPTER - 5

DFD (Data Flow Diagram)



A simple DFD for a movie recommendation system might include:

1. **User Input**: Users submit their preferences and ratings.
2. **Recommendation Engine**: Processes the input data to generate recommendations based on algorithms like collaborative filtering or content-based filtering.
3. **Data Store**: Stores user profiles and movie metadata.
4. **Output**: Displays recommended movies back to the user.

CHAPTER - 6

Structure of Database

A Recommendation System is designed to suggest relevant items (e.g., products, movies, books) to users based on their preferences, interactions, and behaviour. The system can use different recommendation techniques such as collaborative filtering, content-based filtering, or hybrid methods to generate personalized recommendations.

To store and manage the required data efficiently, we need a well-structured database that captures user interactions, item details, and feedback. The following schema provides a relational database structure for implementing a recommendation system.

Database Components:

- **Users Table** – Stores user information such as name, email, age, and location.
- **Items Table** – Stores information about the products, movies, or content that users interact with.
- **User Interactions Table** – Records user actions such as viewing, clicking, purchasing, or liking an item.
- **Ratings Table** – Stores user ratings and reviews for different items.
- **Recommendations Table** – Stores personalized recommendations generated for each user along with a recommendation score.
- **Feedback Table** – Captures user feedback on recommendations to improve future suggestions.

Dataset Link:https://github.com/madesh6554/Smart-Electronics-Recommendation-System/blob/main/Dataset_preprocessed_with_accessory.csv

Use Case Example

1. A **user** logs into an e-commerce platform.
2. The system tracks the user's activity (e.g., viewing a mobile phone).
3. The **Recommendation System** suggests similar products based on the user's history.
4. The user **rates** a product or **provides feedback** on the recommendations.
5. The system **learns from user behaviour** and refines future recommendations.

Data Collection:

Web scraping is a technique used to extract data from websites. It is particularly useful for gathering product details from e-commerce platforms, including product ID, name, description, price, rating, review count, colour, brand, category, and accessories. This documentation outlines the process of web scraping using Python libraries such as requests and BeautifulSoup to collect and structure product-related data.

Prerequisites: To perform web scraping effectively, we need:

- A basic understanding of Python programming.
- Knowledge of HTML and CSS selectors for identifying webpage elements.
- Awareness of ethical considerations and website policies regarding data scraping.

Process Overview:

Accessing the Website: The first step in web scraping involves accessing the target website and retrieving its content. This requires sending an HTTP request to the webpage containing the desired product data.

Parsing the Content: Once the webpage content is obtained, it needs to be parsed to extract meaningful information. This is done by analysing the HTML structure and identifying relevant tags and attributes.

Extracting Product Data: Key product details such as product ID, name, description, price, rating, review count, colour, brand, category, and accessories are identified and extracted from the parsed HTML.

Organizing the Data: The extracted data is structured and stored in an appropriate format, such as a table, spreadsheet, or database, for further analysis and usage.

- Using this dataset to create the Recommendation system based on the Model to choose to create the RS Model.
- After to create the RS model to deploy the model and to do the user interaction and improve these interactions.

Recommendation system:

A recommendation system is a data-driven approach used to suggest relevant products to users based on various criteria. It enhances user experience by recommending similar products, complementary products, and frequently bought products. This documentation outlines the key concepts and methodologies for building an effective recommendation system.

Types of Product Recommendations:

1. **Similar Products:** These are items that share attributes with the currently viewed product, such as brand, category, price range, or user reviews.
2. **Complementary Products:** These are products that pair well with the main item, such as accessories or add-ons that enhance its usability.
3. **Frequently Bought Together:** These recommendations are based on historical purchase patterns, suggesting products commonly purchased in conjunction with the main item.

Methods for Recommendation:

- **Content-Based Filtering:** Uses product attributes like name, category, and features to recommend similar products by comparing their descriptions and characteristics with user preferences.
- **Rule-Based Filtering:** Uses predefined business rules and logic to recommend products based on specific conditions, such as recommending accessories for a purchased product or suggesting seasonal items.

- **Collaborative Filtering:** Analyzes user behavior and preferences to identify patterns and suggest items based on past interactions.
- **Association Rule Mining:** Identifies frequently bought products by analyzing transaction data, such as "customers who bought this also bought that."
- **Hybrid Approach:** Combines multiple recommendation techniques for better accuracy and user personalization.

Data Requirements: To build a recommendation system, the following data points are necessary:

- Product details: ID, name, category, description, and features.
- User interactions: Browsing history, clicks, and ratings.
- Purchase data: Transaction history and frequently bought products.
- Business rules: Predefined logic for rule-based recommendations.

Implementation Process:

1. **Data Collection:** Gather structured product and transaction data from e-commerce platforms.
2. **Data Processing:** Clean and preprocess data to extract meaningful insights.
3. **Model Selection:** Choose an appropriate recommendation model based on business goals and available data.
4. **Recommendation Generation:** Apply filtering techniques such as content-based, rule-based, and collaborative filtering to suggest relevant products.
5. **Evaluation and Optimization:** Continuously monitor the system's performance and refine algorithms for improved recommendations.

Deployment:

Streamlit is an open-source Python framework for building interactive web applications with minimal effort. It's widely used for data science, machine learning, and visualization projects because of its simplicity.

🔹 **Key Features of Streamlit**

✅ **Easy to Use** – No need for front-end experience; just write Python code.

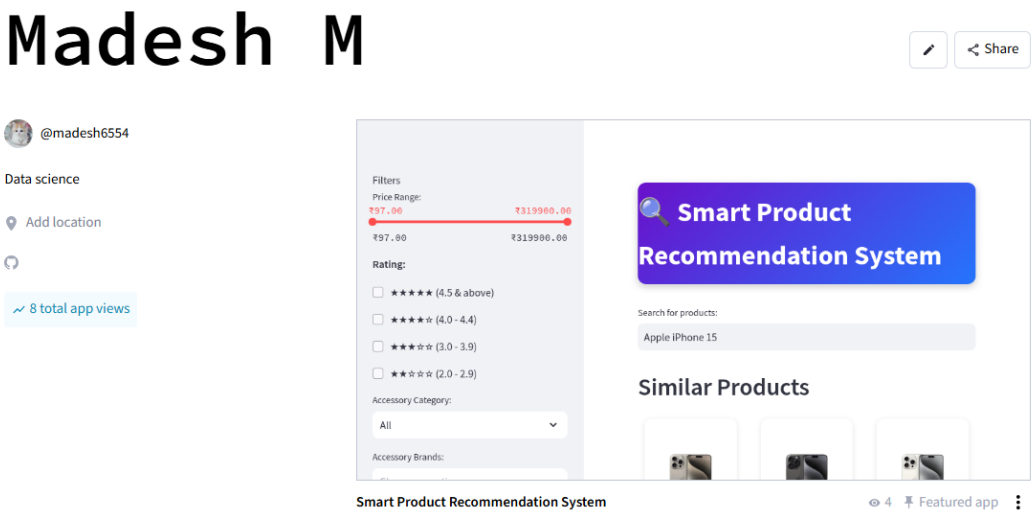
✅ **Interactive Widgets** – Supports buttons, sliders, text inputs, and more.

✅ **Live Updates** – Automatically updates the app when you change the code.

✅ **Integration** – Works well with Pandas, Matplotlib, Plotly, OpenCV, TensorFlow, and PyTorch.

✅ **Deployment** – Can be deployed on Streamlit Cloud, AWS, GCP, or Heroku.

Project website link: https://smart-electronics-recommendation-system



CHAPTER - 7

Advantages vs Limitations

S.No	Advantages	Limitations
1.	User-friendly interface with Streamlit	Slower performance on very large datasets
2.	Smart product recommendations using TF-IDF & cosine similarity	Cold start problem for new or rarely rated products
3.	Handles misspellings using fuzzy matching	Not personalized to individual users
4.	Accessory filtering by category and brand	No real-time updates — manual refresh needed
5.	Flexible filtering options (price, rating, category, brand)	Based on simple text similarity, not semantic meaning
6.	Visually appealing product cards with images	Limited understanding of deeper product relationships
7.	Preprocessing ensures clean, consistent data	No support for multilingual product names or descriptions
8.	Designed to get products only for specific brands like Apple and Samsung	Supports only Apple and Samsung products — not generalizable to other brands
9.	Lightweight and fast to deploy for small-scale applications	Works with a relatively small dataset (e.g., 1500 entries) — may lack coverage

CHAPTER - 8

Sources Code

Data Collection:

```
import streamlit as st
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from fuzzywuzzy import process
import ast


# Set page config at the top
st.set_page_config(layout="wide", page_title="Smart Product Recommendation System")


# Load and preprocess data
@st.cache_data
def load_data():
    df = pd.read_csv("Dataset_preprocessed_with_accessory.csv")
# Data Preprocessing:
    # Data cleaning
    df['price'] = pd.to_numeric(df['price'].replace('Not Available', np.nan), errors='coerce')
    df['rating'] = pd.to_numeric(df['rating'], errors='coerce')
    df['review_count'] = pd.to_numeric(df['review_count'], errors='coerce')
    # Handle missing values
    median_price = df['price'].median()
    df['price'] = df['price'].fillna(median_price)
```

```
    df['rating'] = df['rating'].fillna(0)
    df['review_count'] = df['review_count'].fillna(0)

    # Clean feature lists
    def safe_convert(x):
        try:
            if isinstance(x, str):
                if x.startswith('[') and x.endswith(']'):
                    return ast.literal_eval(x)
                else:
                    return [item.strip() for item in x.split(',')]
            return []
        except (ValueError, SyntaxError):
            return []
    df['features'] = df['features'].apply(safe_convert)
    return df
df = load_data()

# Feature engineering
df['text_features'] = df['product_title'] + ' ' + df['category'] + ' ' + df['brand'] + ' ' +
df['features'].apply(lambda x: ' '.join(x))
tfidf = TfidfVectorizer(stop_words='english', ngram_range=(1,2))
tfidf_matrix = tfidf.fit_transform(df['text_features'])
cosine_sim = cosine_similarity(tfidf_matrix)
```

```
# Recommendation function

def get_recommendations(product_name, df, cosine_sim, price_range, for_accessories=False,
brand_filter=None, category_filter=None, rating_filter=None):

    matches = process.extractBests(product_name.lower(), df['product_title'].str.lower(), limit=1)

    if not matches:

        return []

    idx = df[df['product_title'].str.lower() == matches[0][0].lower()].index[0]

    sim_scores = list(enumerate(cosine_sim[idx]))

    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)[1:]

    seen_products = set()

    recommendations = []

    for i in sim_scores:

        product = df.iloc[i[0]]

        if len(recommendations) >= 5:

            break

# Rule based filters:

        # Price filter

        if not (price_range[0] <= product['price'] <= price_range[1]):

            continue

        # Rating filter

        if rating_filter:

            valid_rating = False

            for (r_min, r_max) in rating_filter:

                if r_min <= product['rating'] < r_max:

                    valid_rating = True

                    break

            if not valid_rating:
```

```
        continue
# Accessory filters
if for_accessories:
    if brand_filter and product['brand'] not in brand_filter:
        continue
    if category_filter != 'All' and product['category'] != category_filter:
        continue
    if not product['is_accessory']:
        continue
else:
    if product['is_accessory']:
        continue
product_sig = f"{product['brand']}-{product['product_title']}"
if product_sig in seen_products:
    continue
seen_products.add(product_sig)
recommendations.append({
    "title": product['product_title'],
    "price": product['price'],
    "brand": product['brand'],
    "rating": product['rating'],
    "review_count": product['review_count'],
    "image": product['image_url'],
    "features": product['features']
})
```

```
 return recommendations
# Streamlit UI
st.markdown("""
    <style>
        .title-container {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 150px;
            background: linear-gradient(135deg, #6a11cb, #2575fc);
            color: #fff;
            font-size: 40px;
            font-weight: bold;
            border-radius: 12px;
            box-shadow: 0 4px 12px rgba(0,0,0,0.2);
            margin-bottom: 30px;
        }
    </style>
    <div class="title-container">
        🔍 Smart Product Recommendation System
    </div>
""", unsafe_allow_html=True)

# Sidebar Filters
with st.sidebar:
    st.markdown('<div class="filter-header">Filters</div>', unsafe_allow_html=True)
```

```
# Price Filter
min_price = float(df['price'].min())
max_price = float(df['price'].max())
price_range = st.slider(
    "Price Range:",
    min_value=min_price,
    max_value=max_price,
    value=(min_price, max_price),
    format="₹%.2f"
)
# Rating Filter
st.markdown("**Rating:**")
rating_5 = st.checkbox("★★★★★ (4.5 & above)")
rating_4 = st.checkbox("★★★★☆ (4.0 - 4.4)")
rating_3 = st.checkbox("★★★☆☆ (3.0 - 3.9)")
rating_2 = st.checkbox("★★☆☆☆ (2.0 - 2.9)")

selected_ratings = []
if rating_5:
    selected_ratings.append((4.5, 5.0))
if rating_4:
    selected_ratings.append((4.0, 4.5))
if rating_3:
    selected_ratings.append((3.0, 4.0))
if rating_2:
    selected_ratings.append((2.0, 3.0))
```

```
    # Category Filter
    accessory_categories = ['All'] + sorted(df[df['is_accessory']]['category'].unique().tolist())
    selected_category = st.selectbox(
        "Accessory Category:",
        options=accessory_categories,
        index=0
    )
    # Brand Filter
    if selected_category == 'All':
        available_brands = df[df['is_accessory']]['brand'].unique().tolist()
    else:
        available_brands = df[(df['is_accessory']) & (df['category'] ==
selected_category)]['brand'].unique().tolist()

    selected_brands = st.multiselect(
        "Accessory Brands:",
        options=sorted(available_brands),
        default=[]
    )

# Main Content
search_query = st.text_input("Search for products:", "Apple iPhone 15")

# Get Recommendations
similar_products = get_recommendations(search_query, df, cosine_sim, price_range,
rating_filter=selected_ratings)
accessories = get_recommendations(search_query, df, cosine_sim, price_range,
```

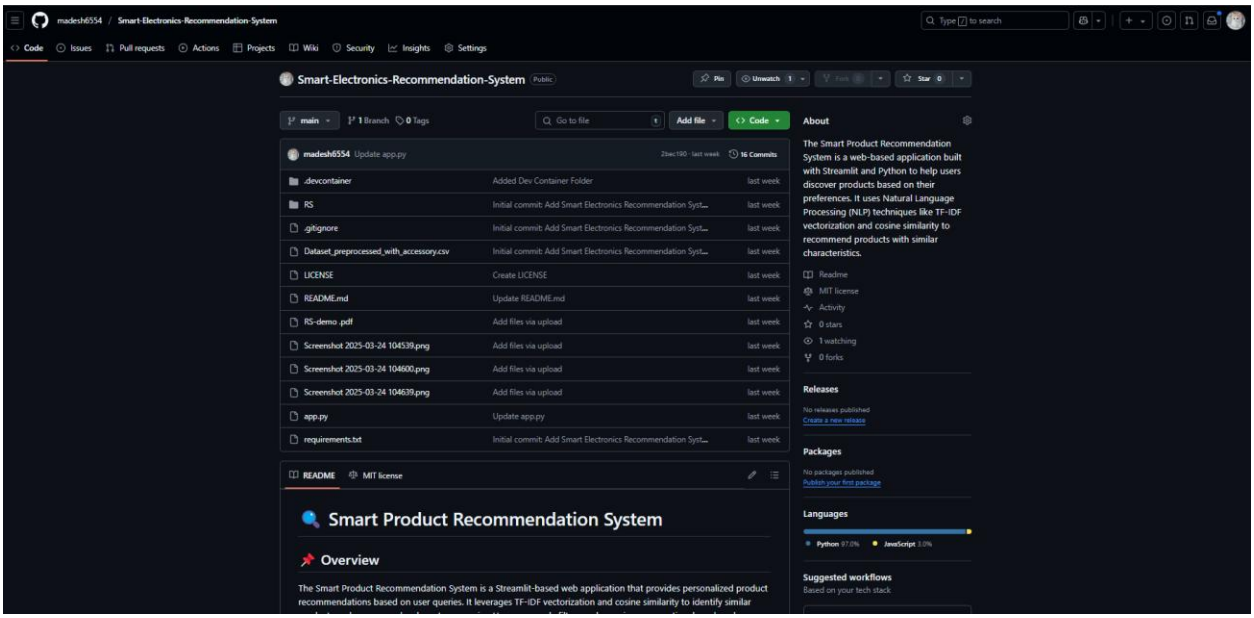
```
                    for_accessories=True,
                    brand_filter=selected_brands,
                    category_filter=selected_category,
                    rating_filter=selected_ratings)

def product_card(product):
    # Ensure features list is properly formatted
    features = product['features'] if isinstance(product['features'], list) else []

    features_html = "".join([f"<li>{feat}</li>" for feat in features]) if features else "<li>No features available</li>"
    return f"""
    <div style="
        padding: 15px;
        border-radius: 10px;
        margin: 10px;
        background: #ffffff;
        box-shadow: 0 2px 8px rgba(0,0,0,0.1);
        color: #333333;
        height: 500px;
        overflow: hidden;
        display: flex;
        flex-direction: column;
    ">
        <img src="{product['image']}" style="border-radius:8px; margin-bottom:10px; width:100%; height:200px; object-fit: contain;">
        <div style="flex-grow: 1; overflow: hidden;">
```

```
        <h4 style="margin:0 0 10px 0; color: #222222; text-align: center; font-size: 1.1em;">{product['title'][:80]}</h4>

        <div style="margin-bottom: 10px; text-align: center;">

            <span style="background: #f0f2f6; padding: 3px 8px; border-radius: 5px; margin-right: 5px;">

                ★ {product['rating']:.1f} ({product['review_count']} reviews)

            </span>

            <span style="background: #e3f2fd; padding: 3px 8px; border-radius: 5px;">

                ₹{product['price']:,.2f}

            </span>

        </div>

        <div style="margin-top: 10px;">

            <b style="color: #1a73e8;">Key Features:</b>

            <div style="max-height: 150px; overflow-y: auto; margin-top: 5px; border: 1px solid #eee; border-radius: 8px; padding: 8px;">

                <ul style="margin: 0; padding-left: 20px; font-size: 14px;">

                    {features_html}

                </ul>

            </div>

        </div>

    </div>

  </div>

  """

def product_details(product):

  with st.expander(f"Details for {product['title']}"):

    st.image(product['image'], caption=product['title'], width=300)  # Use width instead of use_column_width

    st.write(f"**Brand:** {product['brand']}")
```

```
        st.write(f"**Rating:** {product['rating']} (from {product['review_count']} reviews)")
        st.write(f"**Price:** ₹{product['price']:,.2f}")
        st.write("**Features:**")
        for feature in product['features']:
            st.write(f"- {feature}")

# Display Similar Products (Main Products)
if similar_products:
    st.markdown("## Similar Products")
    cols = st.columns(3)
    for idx, product in enumerate(similar_products[:3]):
        with cols[idx % 3]:
            st.markdown(product_card(product), unsafe_allow_html=True)
            product_details(product)
# Display Accessories
if accessories:
    st.markdown("## Recommended Accessories")
    cols = st.columns(4)
    for idx, product in enumerate(accessories[:4]):
        with cols[idx % 4]:
            st.markdown(product_card(product), unsafe_allow_html=True)
            product_details(product)
# Empty state handling
if not similar_products and not accessories:
    st.warning("No products found matching your search. Try different keywords!")
elif not similar_products:
```

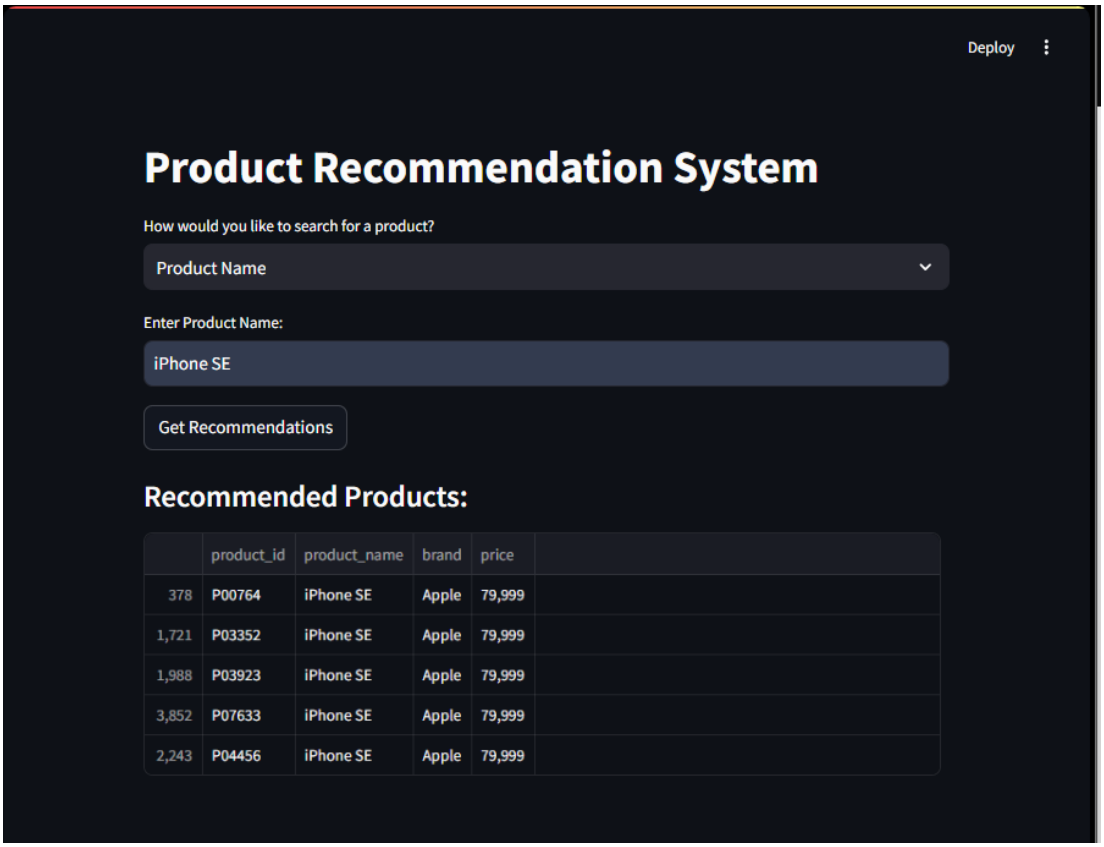
```
    st.warning("No similar products found. Try adjusting your search!")
elif not accessories:
    st.warning("No accessories found matching filters. Try adjusting filters!")


!streamlit run app1.py
```

Project GitHub Repo Link: https://github.com/madesh6554/Smart-Electronics-Recommendation-System



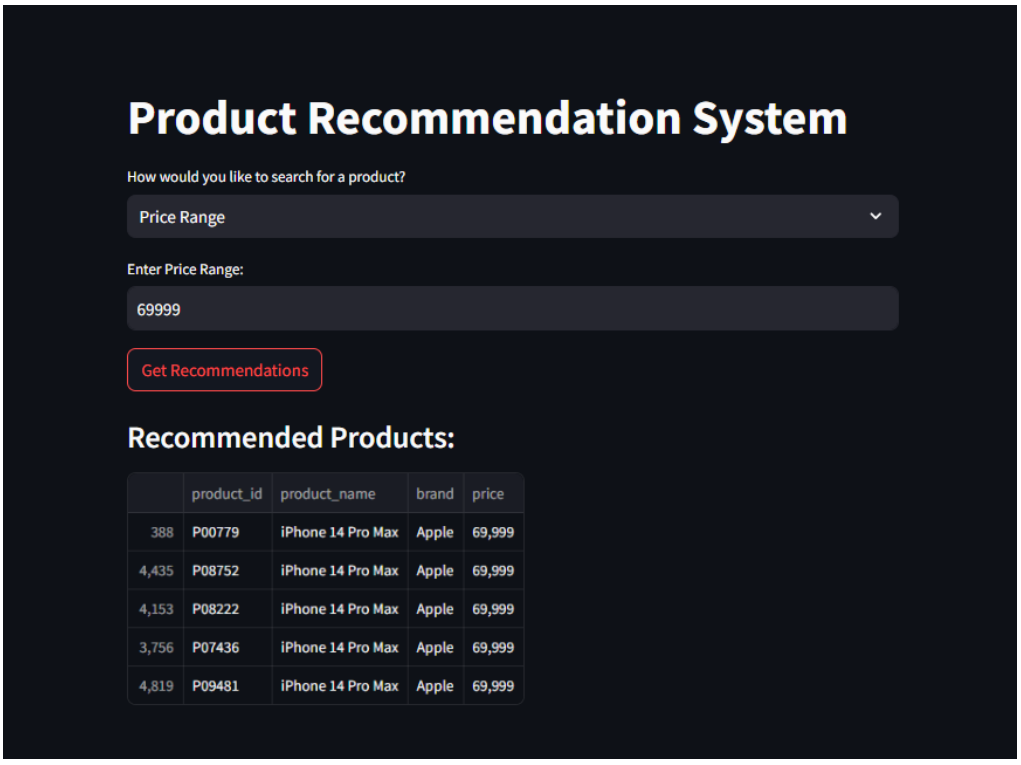
Sample Outputs

Output 1



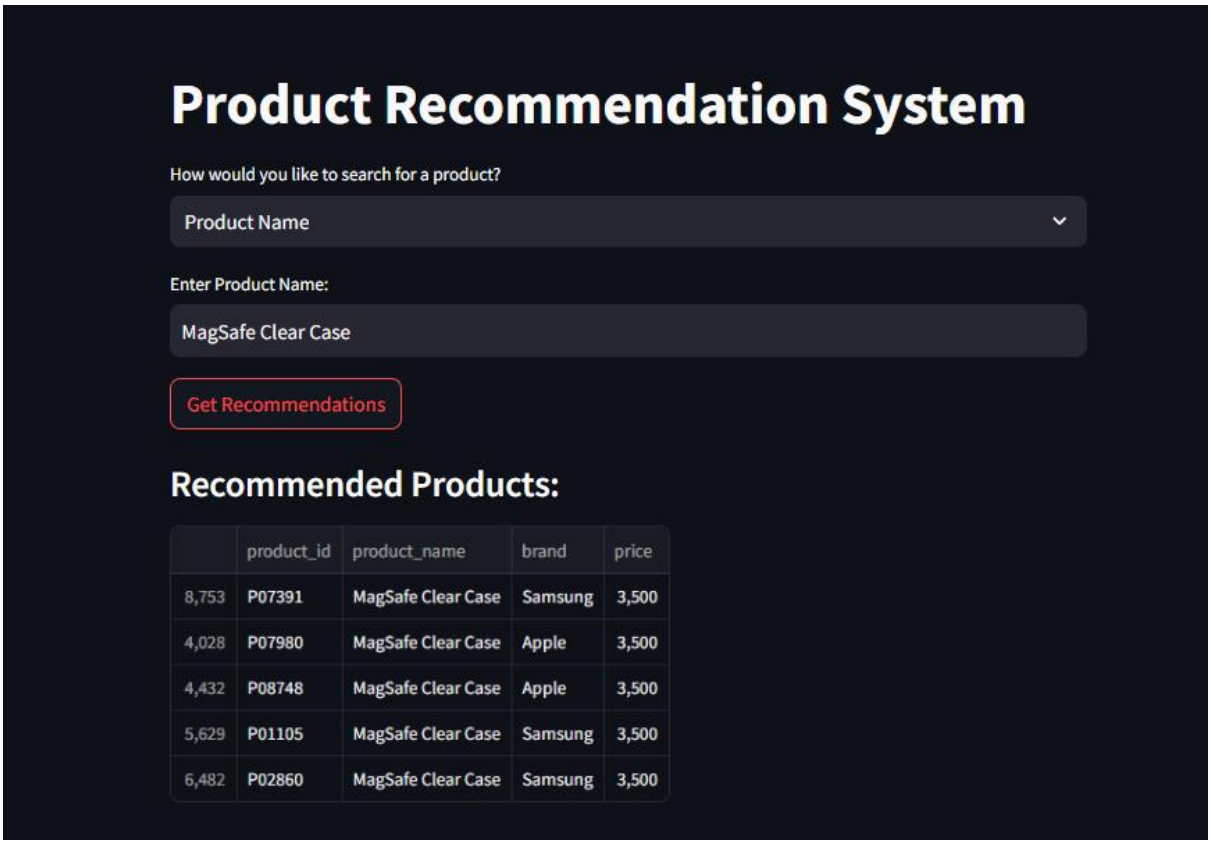
	product_id	product_name	brand	price
378	P00764	iPhone SE	Apple	79,999
1,721	P03352	iPhone SE	Apple	79,999
1,988	P03923	iPhone SE	Apple	79,999
3,852	P07633	iPhone SE	Apple	79,999
2,243	P04456	iPhone SE	Apple	79,999

Output 2:

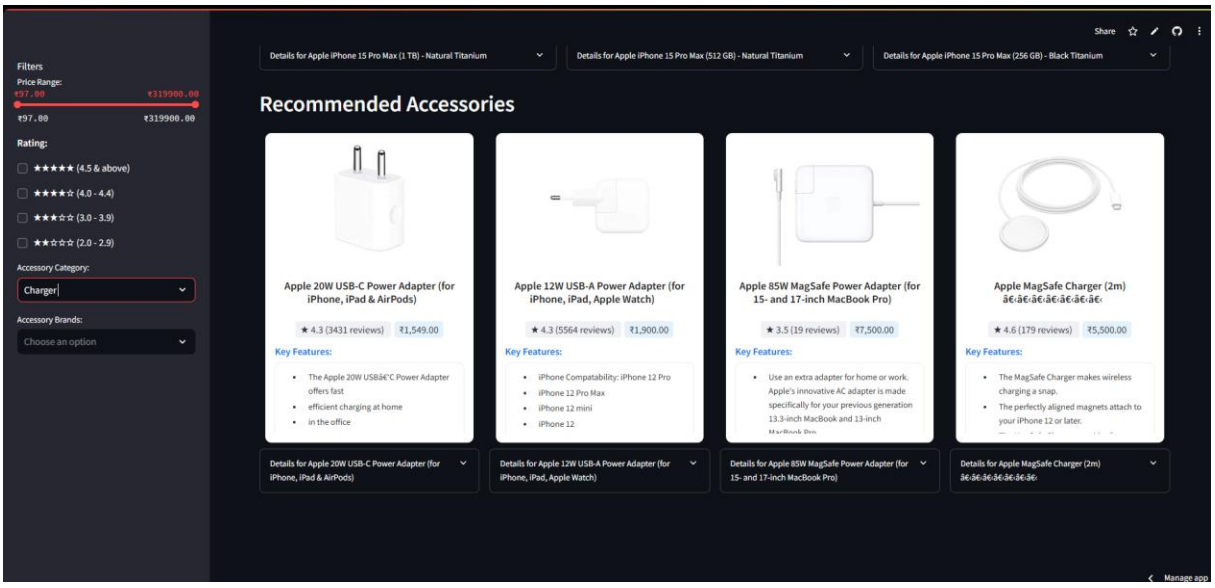


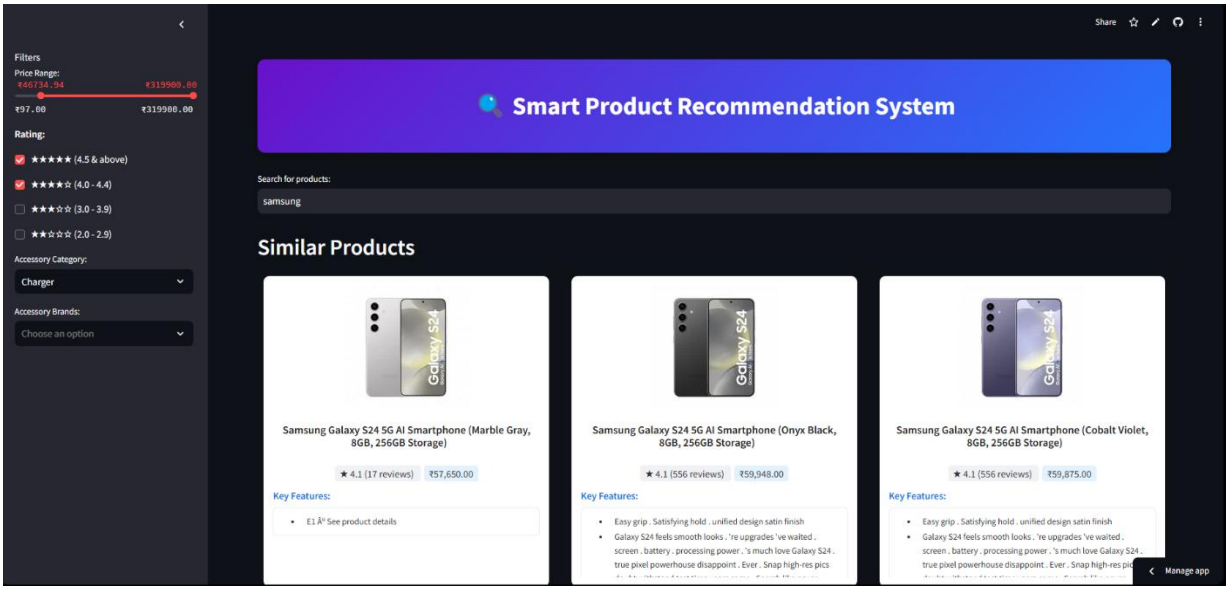
	product_id	product_name	brand	price
388	P00779	iPhone 14 Pro Max	Apple	69,999
4,435	P08752	iPhone 14 Pro Max	Apple	69,999
4,153	P08222	iPhone 14 Pro Max	Apple	69,999
3,756	P07436	iPhone 14 Pro Max	Apple	69,999
4,819	P09481	iPhone 14 Pro Max	Apple	69,999

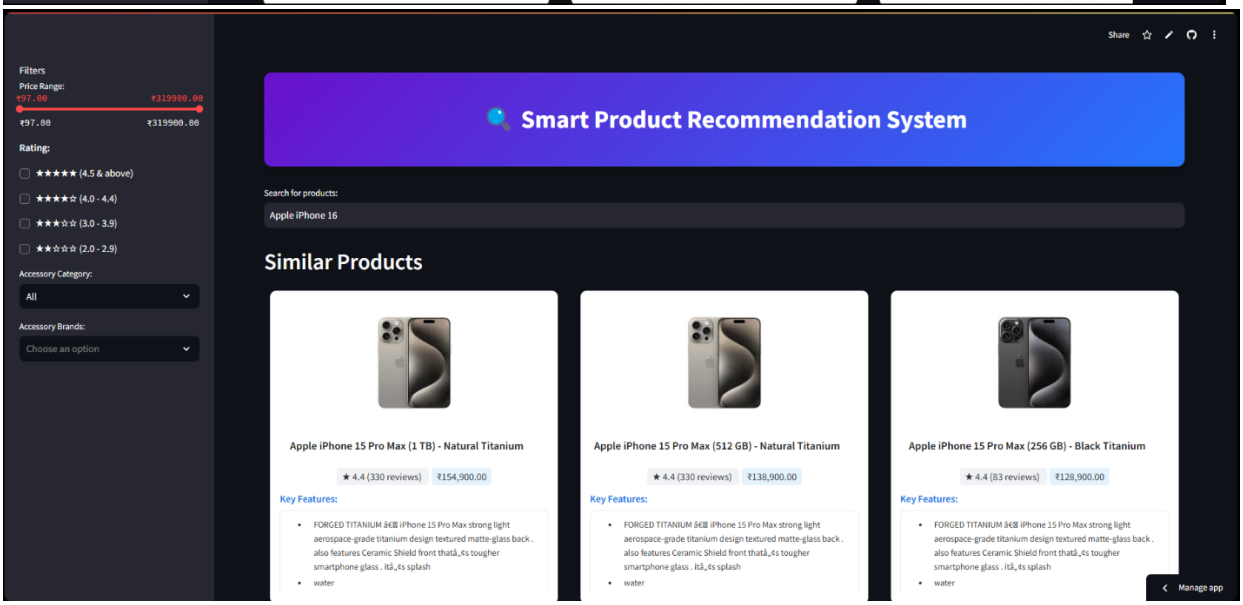
Output 3:



Deployment in Streamlit Cloud:







Project website link: https://smart-electronics-recommendation-system

CONCLUSION

The Smart Product Recommendation System effectively integrates content-based filtering with rule-based filtering to provide personalized and relevant product suggestions. By leveraging TF-IDF vectorization and cosine similarity, the system identifies similar products based on textual attributes, while rule-based constraints ensure that recommendations align with user preferences such as price, rating, brand, and category. This hybrid approach enhances recommendation accuracy, diversity, and user satisfaction. The system is scalable and adaptable, making it suitable for various e-commerce applications. Future improvements could include integrating collaborative filtering and deep learning models to further refine recommendations and enhance user experience.

References

1. Aggarwal, C. C. (2016). *Recommender systems: The textbook*. Springer. https://doi.org/10.1007/978-3-319-29659-3
2. Ricci, F., Rokach, L., & Shapira, B. (Eds.). (2015). *Recommender systems handbook* (2nd ed.). Springer. https://doi.org/10.1007/978-1-4899-7637-6
3. Ramos, J. (2003). Using TF-IDF to determine word relevance in document queries. *Proceedings of the First Instructional Conference on Machine Learning*. https://www.cs.rutgers.edu/~mlittman/courses/ml03/iCML03.pdf
4. Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management, 24*(5), 513–523. https://doi.org/10.1016/0306-4573(88)90021-0
5. Lops, P., de Gemmis, M., & Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. In *Recommender systems handbook* (pp. 73-105). Springer. https://doi.org/10.1007/978-0-387-85820-3_3
6. Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters, 27*(8), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010
7. Cover, T. M., & Thomas, J. A. (2006). *Elements of information theory* (2nd ed.). Wiley. https://doi.org/10.1002/047174882X
8. Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th International Conference on World Wide Web*, 285–295. https://doi.org/10.1145/371920.372071
9. Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 43–52. https://doi.org/10.5555/2074094.2074100
10. McNee, S. M., Riedl, J., & Konstan, J. A. (2006). Being accurate is not enough: How accuracy metrics have hurt recommender systems. *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, 1097–1101. https://doi.org/10.1145/1125451.1125659
11. Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research, 3*, 993–1022. https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf
12. Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press. https://nlp.stanford.edu/IR-book/

13. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017). Neural collaborative filtering. *Proceedings of the 26th International Conference on World Wide Web*, 173–182. https://doi.org/10.1145/3038912.3052569
14. Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer, 42*(8), 30–37. https://doi.org/10.1109/MC.2009.263
15. Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. *Proceedings of the 8th IEEE International Conference on Data Mining*, 263–272. https://doi.org/10.1109/ICDM.2008.22