Sri Lankan Institute of Information Technology

Information Retrieval and Web Analytics (IT3041)

Team Insight Architects

Personalized Ecommerce Product Recommendation System

Final Report 2024

Group Members

Member	Student ID
Gunathilaka P A S R	IT22136824
Himsara P V D	IT22367112
Kumarage H A	IT22320964
Samaraweera W D U I	IT22258526

GitHub Repository Link - https://github.com/Ranketh2004/Irwa-Proj.git

Declaration

We declare that this project report or part of this was not a copy of a document done by any organization, university or any other institute, previous students project groups at SLIIT and was not copied from the internet or other external sources.

Acknowledgement

We owe a debt of gratitude to our lecturer in charge, Mr. Samadhi Chathuranga Rathnayaka and Dr. Lakmini Abeywardhana for giving us the fundamental knowledge, sharing expertise, giving instructions and coordinating throughout the project.

Table of content

Abstract

This project focuses on developing a personalized product recommendation system for an e-commerce platform using Python Flask, front-end web technologies, and machine learning techniques. The system enhances the shopping experience by helping users discover products they may want to purchase. It incorporates multiple recommendation techniques, including rating-based filtering, content-based filtering, and collaborative filtering. By combining these methods into a hybrid approach, the system delivers more accurate and relevant recommendations, ultimately improving overall performance and user satisfaction.

Objectives:

- **Personalized Recommendations:** To display products tailored to each user's preferences.
- **Rating-Based Filtering:** To recommend products that other users highly rated, using average rating per product by each user.
- **Hybrid Recommendation Approach:** To improve recommendation accuracy by combining content-based and collaborative filtering methods.
- **Web Integration:** To develop a user-friendly web app using Flask, HTML and CSS allowing seamless interaction with the recommendation system.
- **Address Cold Start Problem:** To provide new users with highly rated products and personalized recommendations based on their preferences.
- **Deployment:** Deploy the web application so that users can access the system from their own device.

Methodology:

- **Data Processing:** The system utilizes an e-commerce dataset containing product details such as product name, ratings, category, brand and descriptions. The data is cleaned and preprocessed before being used by the recommendation model.
- **Content-Based Filtering:** This method suggests products like those the user has viewed, based on attributes such as descriptions, tags, or categories.
- **Collaborative Filtering:** Recommendations are generated by analyzing the behavior of users with similar preferences, suggesting products that appeal to these similar user groups.
- **Rating-Based Filtering:** This method recommends products with higher ratings from other users, assuming that products with strong ratings will be of interest to new or existing users.
- **Hybrid Model:** A combination of content-based and collaborative filtering approaches is used to improve the accuracy and precision of the recommendations.

- **System Deployment:** The recommendation system is integrated into a Flask web application, allowing users to receive real-time product recommendations as they browse the website.

Introduction

E-commerce websites represent a point of purchase for today's consumers in the digital world on everything, from fashion to electronic products, just a click away. On the other hand, the huge number of available products often overwhelms the user and makes it difficult for him to find items that match his preferences. Therefore, recommendation systems have been developed as an important tool to deal with this problem on online shopping platforms. Recommendation systems have the power to really enhance the user's shopping experience by suggesting products that a given user will really like based on their previous interactions. This project develops an e-commerce recommendation system with Flask and uses various machine learning techniques to suggest personalized products to users and enhance their engagement and satisfaction.

Background Information

The e-commerce platform is always on the lookout for how to show a more personalized experience to users. Recommendation systems always have a key role to perform in that respect, estimating the types of items users might be interested in according to data such as product attributes, user interactions, and purchase history. There are various kinds of recommendation systems, including content-based models, which recommend products like what a user has liked, and collaborative filtering models, which suggest products based on what similar users have bought or highly rated. This can be further combined using a hybrid approach to generate even more accurate, diverse recommendations. Machine learning provides the tools to develop such systems that allow platforms to tailor their services to the tastes of each user.

Objective and Significance of the Project

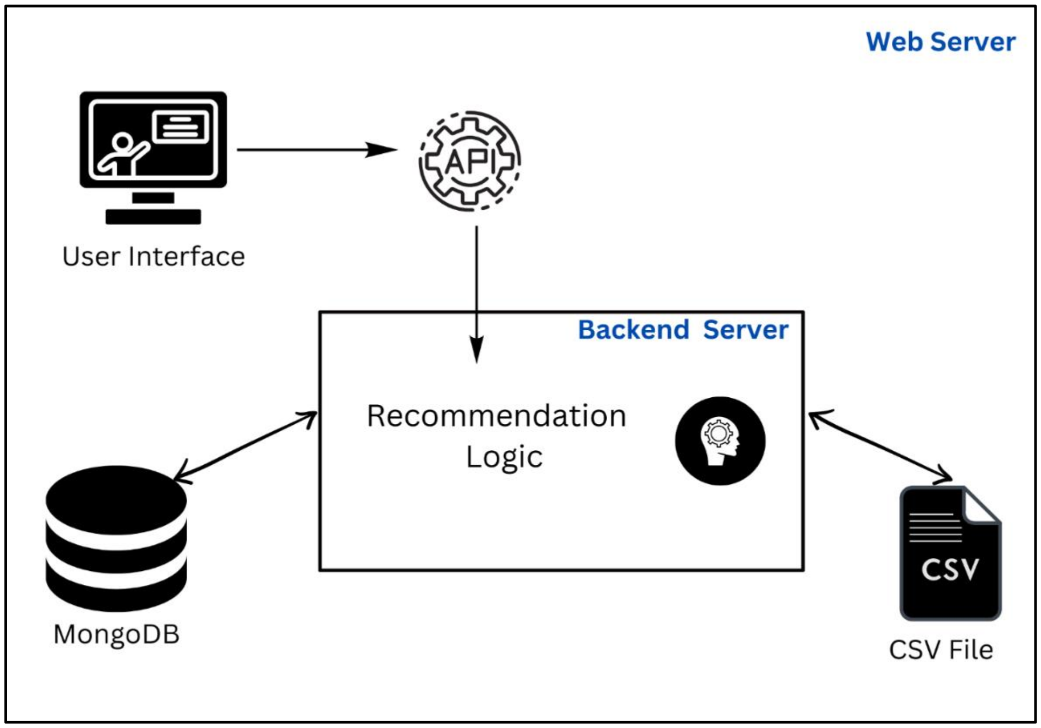
This project is based on the design of an e-commerce website recommendation engine for better product discovery. Machine learning methodologies such as content-based filtering, collaborative filtering, rating based and hybrid models will be implemented to provide personalized recommendations that best match user preference. In addition, it will be integrated with a Flask web application to let the users have a seamless experience: product visualization, personalized recommendations, and user feedback about the recommendation. This project is important, as it enhances the customer's shopping experience; at the same time, on the other side, it increases the capability of the platform to surge sales, increase user retention, and build loyalty through personalization of the offering of products.

Scope and Limitations of the Project

The work will include everything, from collecting and preprocessing the data to implementing different recommendation algorithms and the integration of the system into a web application on top of Flask. In this way, the user will be able to interface with the system by navigating through the products, having a look at the personalized suggestions.

Although the project has some advantages, there are some limitations: for example, the quality of the recommendations depends on available data-if there is less data regarding user interactions and the features of products, then the recommendations would not be very accurate. Furthermore, when the dataset becomes huge, scalability might become an issue because further optimization would be needed to apply it on high-traffic platforms. While machine learning models can give personalized results, the effectiveness of these models is inversely limited to the extent that diverse data will be able to give such results and user engagement with the platform.

System Design and Architecture



Frontend: The HTML, CSS, and JS frontend of e-commerce recommendation system is where users input their name, email address, password and choose their desired categories and brands when creating an account. Once users have subscribed, they are able to search an item , and enter the number of recommendations they require. The recommended items are shown as cards with product images, title, brand, and star rating computed based on the ratings from users. This interactive design gives information updates as the user works through the site, without having to use page reloading to get new data which make it user friendly.

Backend: The backend is developed using Flask in Python. It captures a request from the user, processes it through the recommendation algorithm and could relay the result back to the frontend. It includes the following modules:

- Content-Based Filtering: This recommends items similar to a given item using keyword search and similarity in ideal cosine space with features being TF-IDF of item descriptions.
- User-Based Filtering: This recommends items based on user’s similarity by forming a user-item matrix and calculations carried out by using cosine similarity.
- Hybrid Filtering: Both content-based and user-based approaches are used here, their suggestions are combined, and any repetitions are eliminated.

- Rating Based Filtering: Highest rated products are recommended.

Database:

- Pickled Dataset (data.pkl)

The original dataset was a csv file. After preprocessing, we converted it to a pickle file. Pickled Dataset (data. pkl) itself, it is a small file containing information about items such as names, brands, ratings tags and URL of images. At runtime, this dataset is loaded into memory so that the recommendation system can operate efficiently without needing an external database to store or use recommendations.

- MongoDB

Important user information, for instance- names, emails, passwords and the chosen categories and brands that the user prefers persisted in a MongoDB backend database used by the e-commerce recommendation system. It takes charge in all thing's user authentication, linking at both signup and login times, as well as handling storing user preferences that can improve the personalization recommendation.

Technology Stack

Flask

- Type: Web Framework (Backend)
- Purpose: This MVC class deals with routing, processing of requests and throwing the template. Serves as an intermediary in a client (frontend) and server (recommendation system and MongoDB) arrangement.

HTML/CSS

- Type: Mark-up and Styling Languages of web applications front end.
- Purpose: HTML builds different structures: web pages, forms, and the layout of the recommendation card; CSS refines it by making this layout responsive and user friendly.

JavaScript

- Type: Scripting Language (Frontend web applications only)
- Purpose: Allows for submission of forms in a non-blocking manner and modifies the GUI to smoothly skip between states (login, signup, recommendation).

Scikit-learn

- Type: Machine Learning Library
- Purpose: TF-IDF Vectorizer is used to convert tags or the item descriptions for the content-based filtering and Cosine Similarity is used for measuring the item descriptions (content based) and users (user based).

Pandas

- Type: Data Manipulation Library
- Purpose: Stores data into Data Frame to ease the construction of user-item matrices and extraction of required data columns such as item title and brand, ratings, etc.

Pickle

- Type: Serialization Library
- Purpose: Saves the dataset in a pickled file named `data.pkl` so that recommendations don't require an external database to be stored.

MongoDB

- Type: NoSQL Database (Backend)
- Purpose: Receives information about the users (names, emails, hashed passwords, categories/brands preferences) and handles the processes of signing up / logging in users but does not store recommended data.

Implementation

Backend Development

The backend is developed using **Flask**, a lightweight web framework in Python. Flask provides a simple yet powerful platform to build web apps, enabling efficient routing and handling requests and responses. Flask was chosen for its flexibility and ease of integration with the recommendation logic and data management layer.

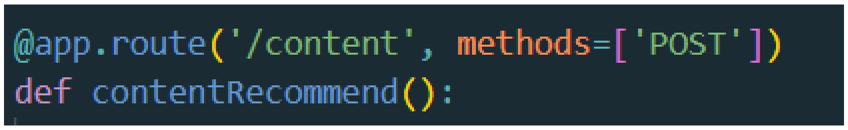
The **recommendation logic** embedded in the backend consists of 5 approaches.

1. User-based filtering -
2. Content-based filtering
3. Rating-based filtering
4. Custom recommendation logic for new users based on their preferences chose
5. Hybrid filtering

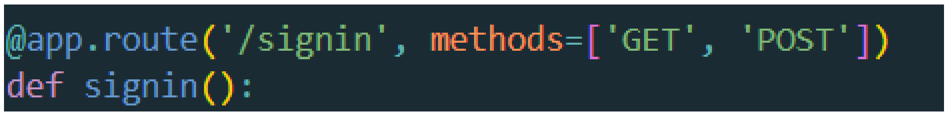
MongoDB serves as the database to store user information such as email, passwords and user preferences such as categories and brands. Flask interacts with MongoDB using a python library called **PyMongo**. When user's submit their preferences, these details are stored in MongoDB and when the system generates recommendations, it retrieves necessary user data to personalize the output.

The product data used for recommendations is loaded from a csv file which is saved as a pickle file into a pandas data frame at runtime. The backend then applies the recommendation logics based on different scenarios. Once the recommendations are generated, they are returned to the frontend for display.

The backend exposes **RESTFUL APIs** that handle user input such as submitting preferences or requesting recommendations. These APIs are implemented as flask routes, allowing communication between frontend and the backend.

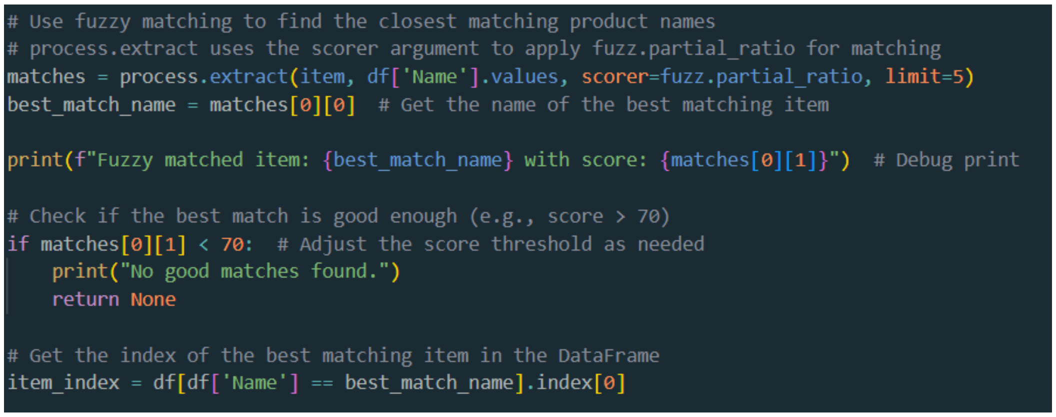


```
@app.route('/content', methods=['POST'])
def contentRecommend():
```



```
@app.route('/signin', methods=['GET', 'POST'])
def signin():
```

Flask manages user sessions to ensure that users can log in, store preferences and retrieve personalized recommendations based on their saved information. The backend also includes error handling mechanisms such as debug print statements to track the flow of data such as the number of filtered data.



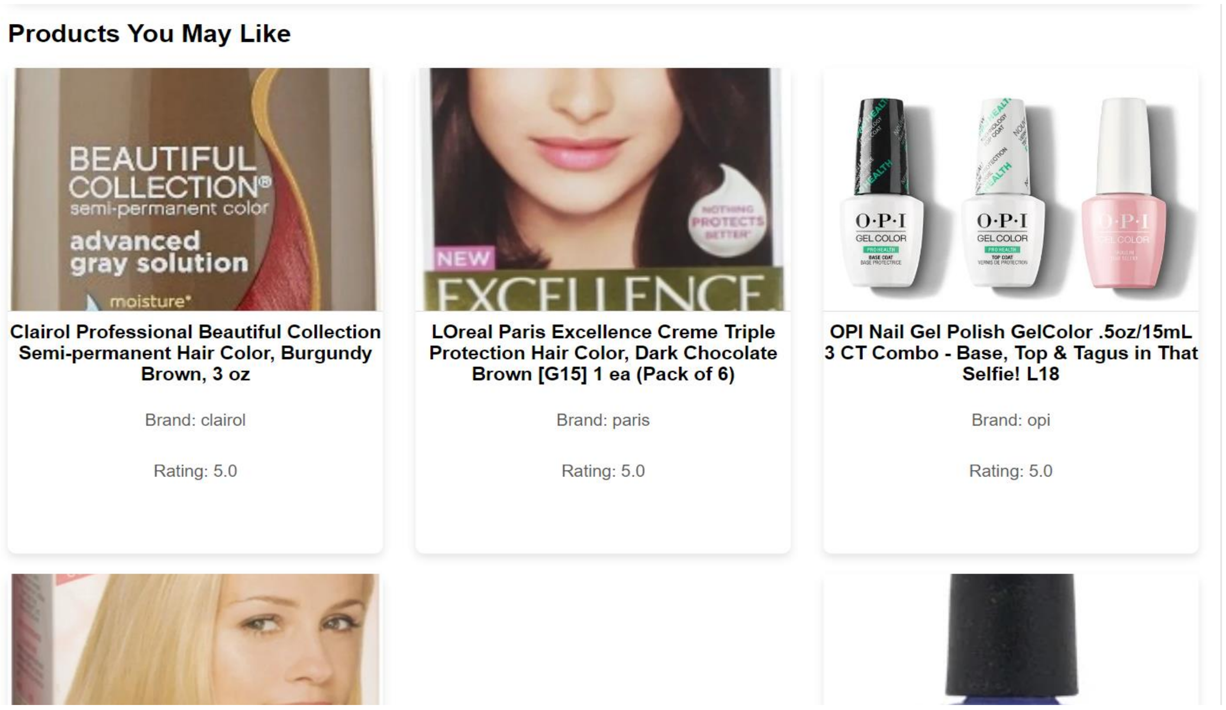
```
# Use fuzzy matching to find the closest matching product names
# process.extract uses the scorer argument to apply fuzz.partial_ratio for matching
matches = process.extract(item, df['Name'].values, scorer=fuzz.partial_ratio, limit=5)
best_match_name = matches[0][0]  # Get the name of the best matching item

print(f"Fuzzy matched item: {best_match_name} with score: {matches[0][1]}")  # Debug print

# Check if the best match is good enough (e.g., score > 70)
if matches[0][1] < 70:  # Adjust the score threshold as needed
    print("No good matches found.")
    return None

# Get the index of the best matching item in the DataFrame
item_index = df[df['Name'] == best_match_name].index[0]
```

Using the user's input, Fuzz from **rapidfuzz** finds the closest matching product name in the dataset. The user does not wish to enter the complete item name, which increases the efficiency of our system. Using fuzz.partial_ratio, it finds the best match and determines whether the match score is higher than a predetermined level (70). The matching product index is obtained if the match appears to be acceptable.

Frontend Development

- **For old users**

  Products recommended for old users are displayed in the **Products You May Like** section



Products You May Like

Clairol Professional Beautiful Collection Semi-permanent Hair Color, Burgundy Brown, 3 oz

Brand: clairol

Rating: 5.0

LOreal Paris Excellence Creme Triple Protection Hair Color, Dark Chocolate Brown [G15] 1 ea (Pack of 6)

Brand: paris

Rating: 5.0

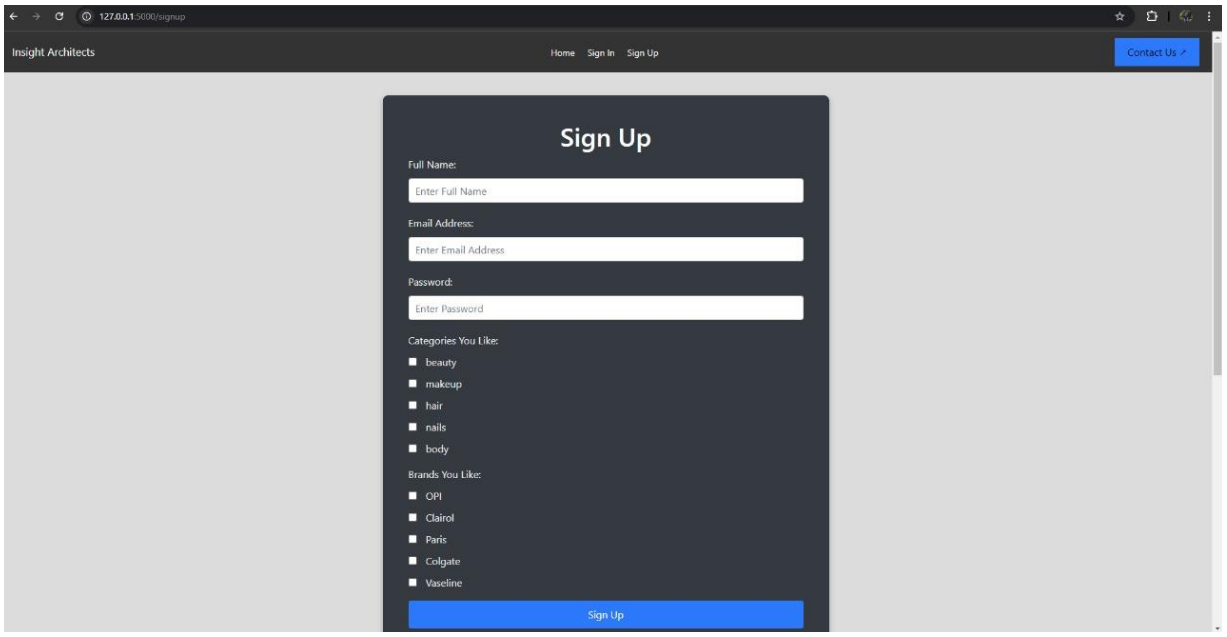
OPI Nail Gel Polish GelColor .5oz/15mL 3 CT Combo - Base, Top & Tagus in That Selfie! L18

Brand: opi

Rating: 5.0

- **For new users**

  The system gathers user data from the sign-up form, including the user's complete name, email address, password, and preferred categories and brands.



127.0.0.1:5000/signup

Insight Architects

Home Sign In Sign Up

Contact Us ↗

Sign Up

Full Name: Enter Full Name

Email Address: Enter Email Address

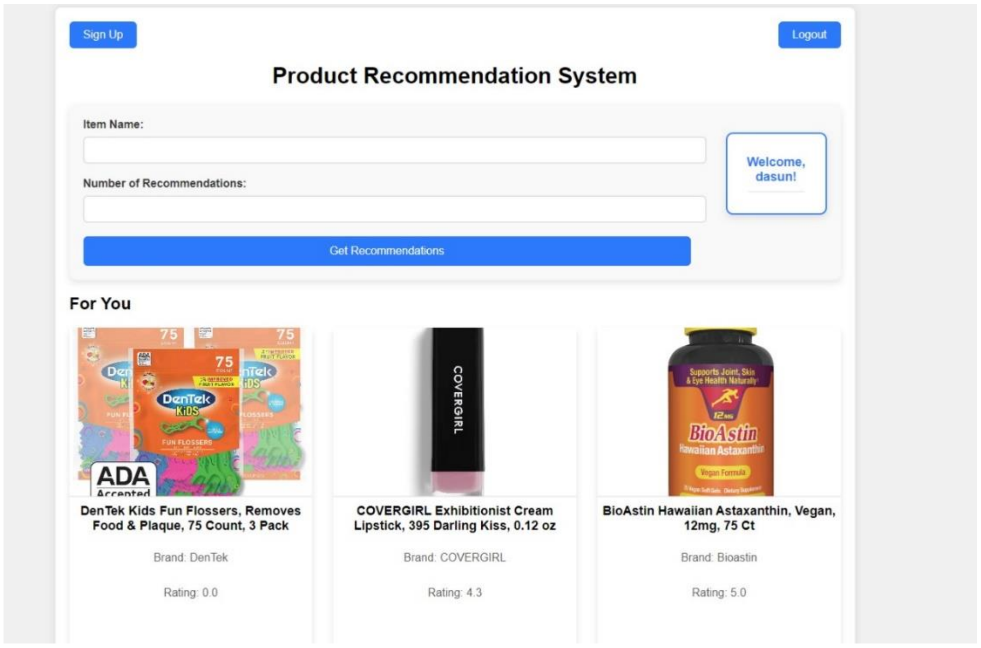
Password: Enter Password

Categories You Like:
- beauty
- makeup
- hair
- nails
- body

Brands You Like:
- OPI
- Clairol
- Paris
- Colgate
- Vaseline

Sign Up

For new users, the recommended products are displayed in the **For You** section based on brands and categories they provided.



Sign Up Logout

Product Recommendation System

Item Name:

Number of Recommendations:

Get Recommendations

Welcome, dasun!

For You

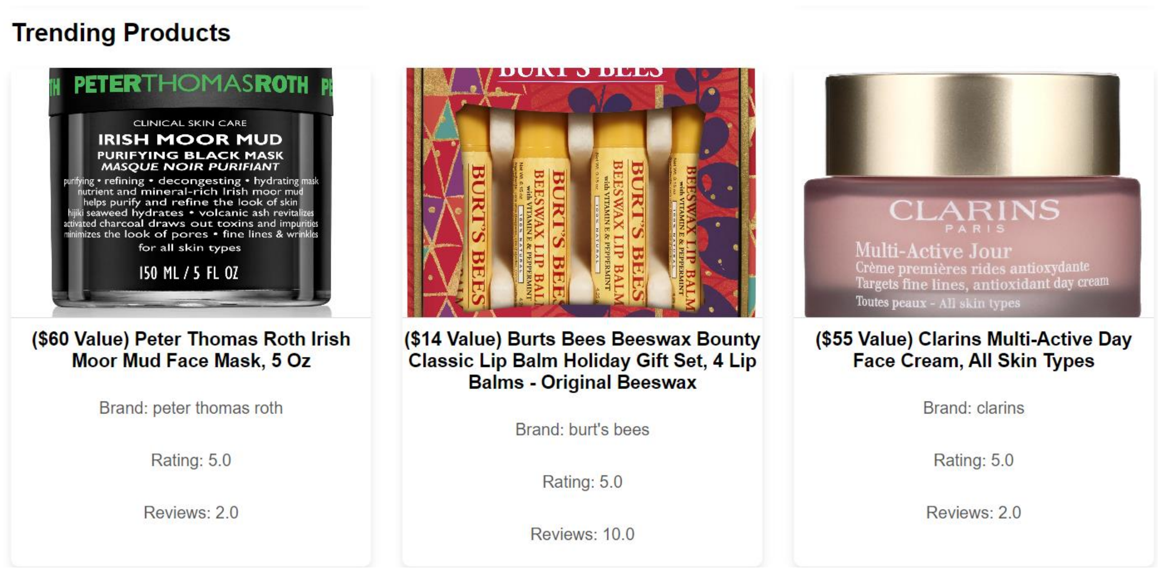
DenTek Kids Fun Flossers, Removes Food & Plaque, 75 Count, 3 Pack
Brand: DenTek
Rating: 0.0

COVERGIRL Exhibitionist Cream Lipstick, 395 Darling Kiss, 0.12 oz
Brand: COVERGIRL
Rating: 4.3

BioAstin Hawaiian Astaxanthin, Vegan, 12mg, 75 Ct
Brand: Bioastin
Rating: 5.0

- **Trending products**

  Trending products are displayed for both old and new users in the **Trending products** section.



Trending Products

($60 Value) Peter Thomas Roth Irish Moor Mud Face Mask, 5 Oz

Brand: peter thomas roth

Rating: 5.0

Reviews: 2.0

($14 Value) Burts Bees Beeswax Bounty Classic Lip Balm Holiday Gift Set, 4 Lip Balms - Original Beeswax

Brand: burt's bees

Rating: 5.0

Reviews: 10.0

($55 Value) Clarins Multi-Active Day Face Cream, All Skin Types

Brand: clarins

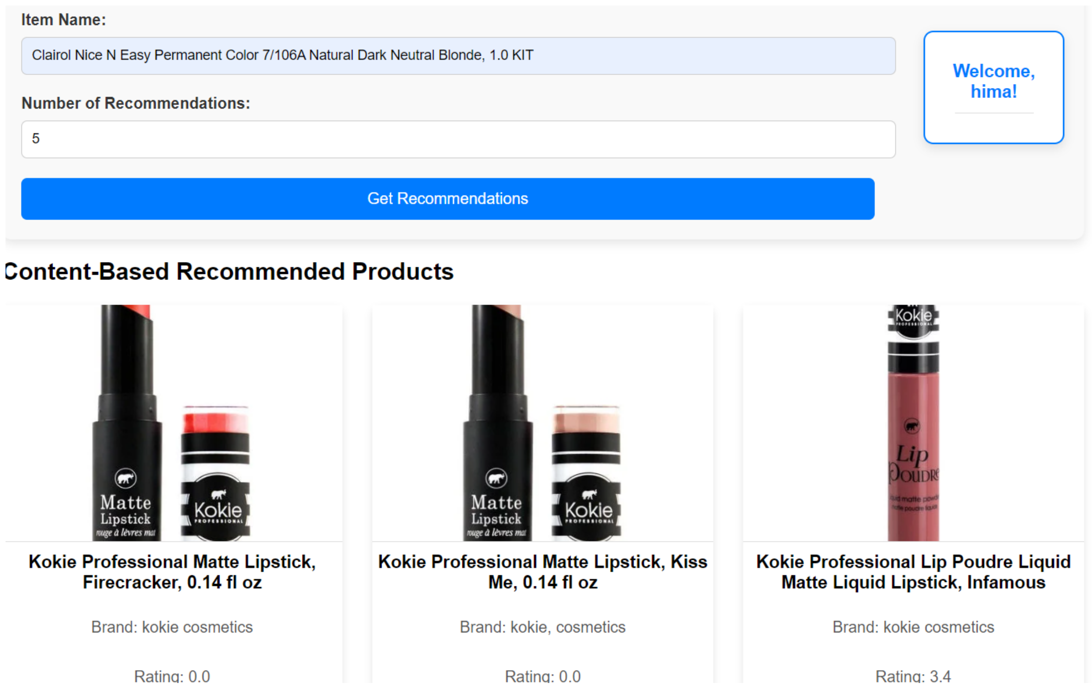
Rating: 5.0

Reviews: 2.0

- **Content based recommended products**

  When a user search a product in the search bar using item name or a key word , the most similar products are recommended.

  For full item name,



Item Name:

Clairol Nice N Easy Permanent Color 7/106A Natural Dark Neutral Blonde, 1.0 KIT

Number of Recommendations:

5

Content-Based Recommended Products

Kokie Professional Matte Lipstick, Firecracker, 0.14 fl oz

Brand: kokie cosmetics

Rating: 0.0

Kokie Professional Matte Lipstick, Kiss Me, 0.14 fl oz

Brand: kokie, cosmetics

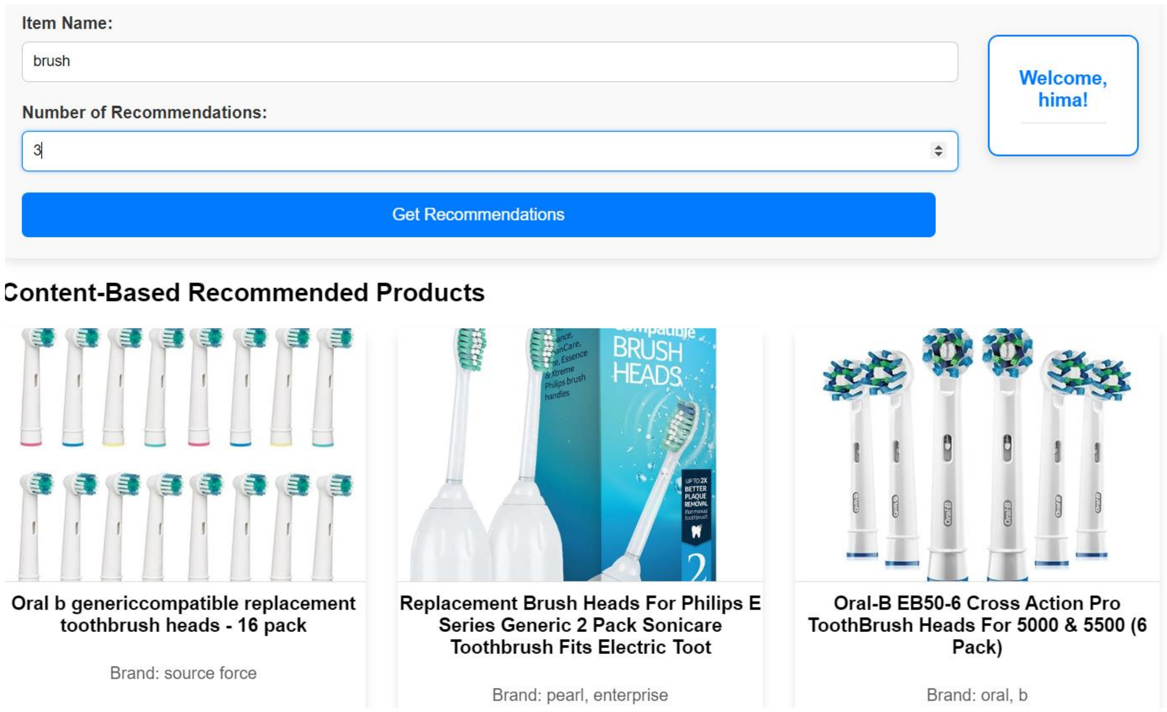
Rating: 0.0

Kokie Professional Lip Poudre Liquid Matte Liquid Lipstick, Infamous

Brand: kokie cosmetics

Rating: 3.4

For key word search,



Content-Based Recommended Products

Oral b genericcompatible replacement toothbrush heads - 16 pack

Brand: source force

Replacement Brush Heads For Philips E Series Generic 2 Pack Sonicare Toothbrush Fits Electric Toot

Brand: pearl, enterprise

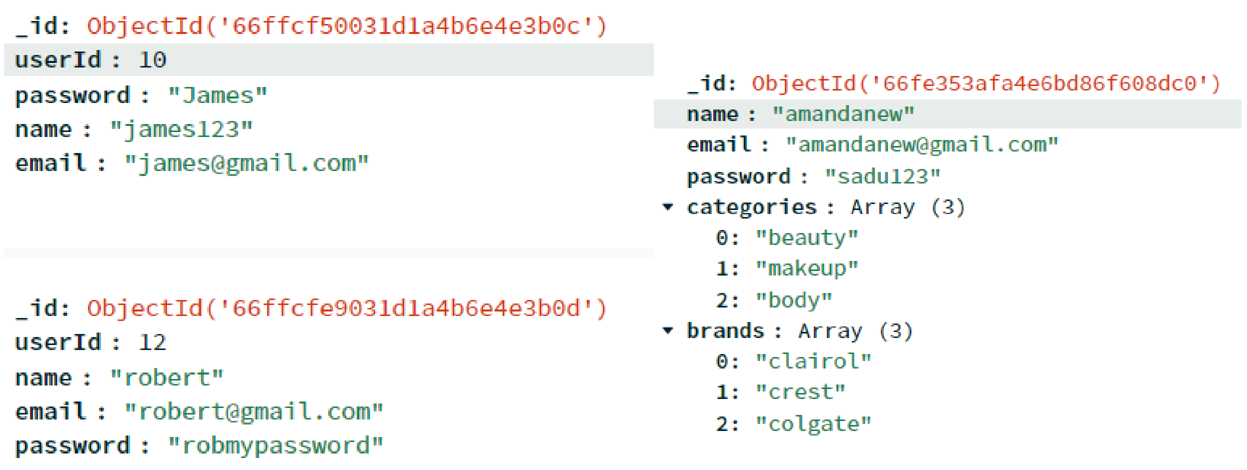
Oral-B EB50-6 Cross Action Pro ToothBrush Heads For 5000 & 5500 (6 Pack)

Brand: oral, b

Database Management

- **Data storage solution**

  MongoDB for user preferences and authentication information : retains user preferences, including brands and categories, as well as user information like name, password, and ID. The selection of Mongo DB is perfect for managing diverse and changing data since it provides simple connectivity to the Flask backend, adaptable schema design, and simple scaling.



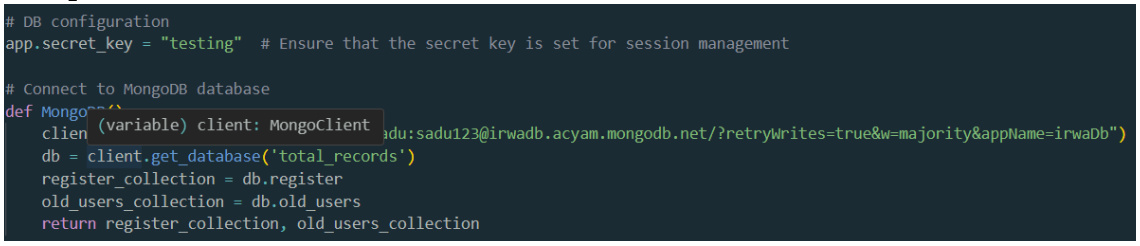
```
_id: ObjectId('66ffcf50031d1a4b6e4e3b0c')
userId : 10
password : "James"
name : "james123"
email : "james@gmail.com"

_id: ObjectId('66ffcfe9031d1a4b6e4e3b0d')
userId : 12
name : "robert"
email : "robert@gmail.com"
password : "robmypassword"
```

```
_id: ObjectId('66fe353afa4e6bd86f608dc0')
name : "amandanew"
email : "amandanew@gmail.com"
password : "sadu123"
▾ categories : Array (3)
    0: "beauty"
    1: "makeup"
    2: "body"
▾ brands : Array (3)
    0: "clairol"
    1: "crest"
    2: "colgate"
```

Products dataset csv file: Product attributes like Category, Brand, Rating, and ImageURL are represented as columns in the CSV file. Processing the data throughout the filtering and recommendation stages is made straightforward by this flat file format.

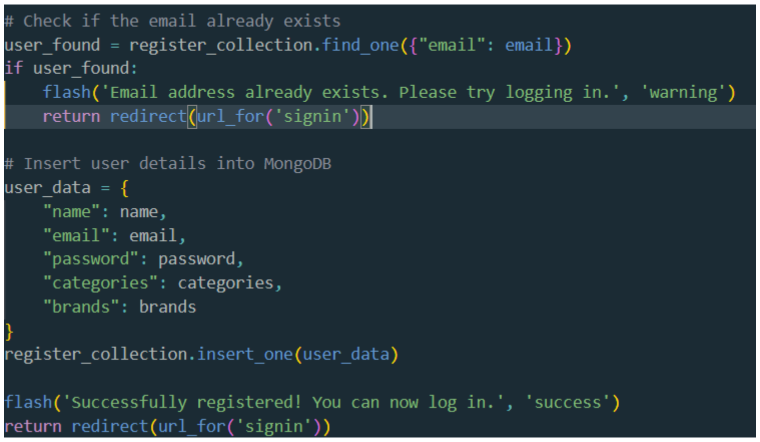
- ## Data Handling and access

Mongo DB connection to the database.



```
# DB configuration
app.secret_key = "testing"  # Ensure that the secret key is set for session management

# Connect to MongoDB database
def Mongo
    clien (variable) client: MongoClient adu:sadu123@irwadb.acyam.mongodb.net/?retryWrites=true&w=majority&appName=irwaDb")
    db = client.get_database('total_records')
    register_collection = db.register
    old_users_collection = db.old_users
    return register_collection, old_users_collection
```

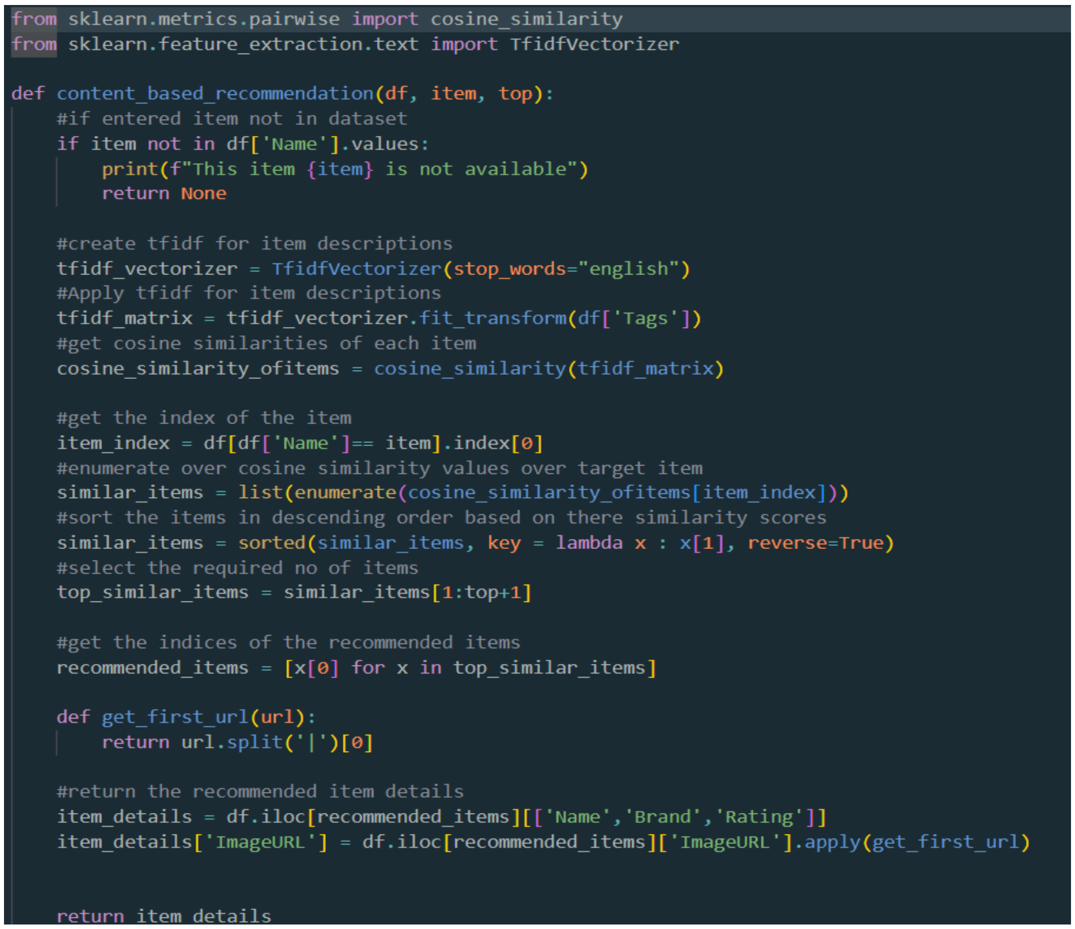
Checks if the user provided email address already exists. If so redirects to sign in page. Otherwise, insert user data to the mongo db collection.



```
# Check if the email already exists
user_found = register_collection.find_one({"email": email})
if user_found:
    flash('Email address already exists. Please try logging in.', 'warning')
    return redirect(url_for('signin'))

# Insert user details into MongoDB
user_data = {
    "name": name,
    "email": email,
    "password": password,
    "categories": categories,
    "brands": brands
}
register_collection.insert_one(user_data)

flash('Successfully registered! You can now log in.', 'success')
return redirect(url_for('signin'))
```

Key Features and Algorithms

- Content Based Filtering



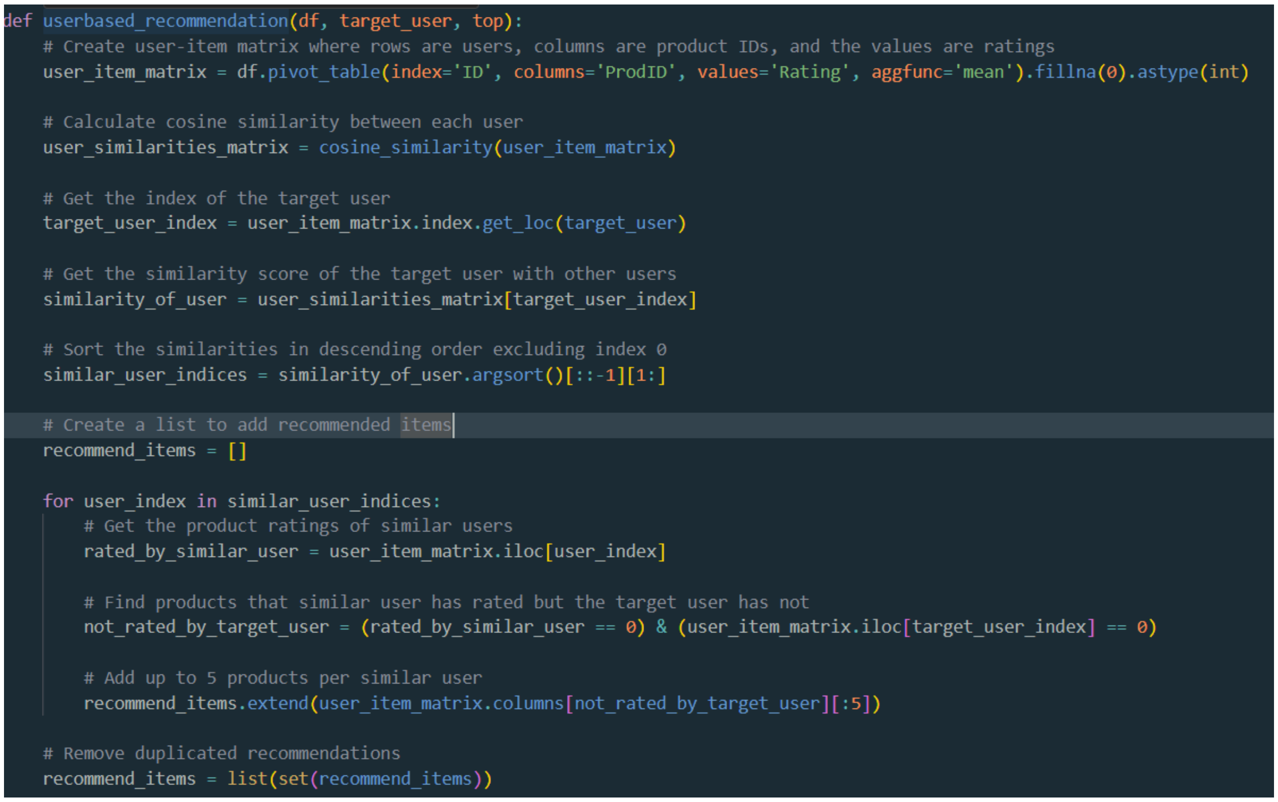
```python
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer

def content_based_recommendation(df, item, top):
    #if entered item not in dataset
    if item not in df['Name'].values:
        print(f"This item {item} is not available")
        return None

    #create tfidf for item descriptions
    tfidf_vectorizer = TfidfVectorizer(stop_words="english")
    #Apply tfidf for item descriptions
    tfidf_matrix = tfidf_vectorizer.fit_transform(df['Tags'])
    #get cosine similarities of each item
    cosine_similarity_ofitems = cosine_similarity(tfidf_matrix)

    #get the index of the item
    item_index = df[df['Name']== item].index[0]
    #enumerate over cosine similarity values over target item
    similar_items = list(enumerate(cosine_similarity_ofitems[item_index]))
    #sort the items in descending order based on there similarity scores
    similar_items = sorted(similar_items, key = lambda x : x[1], reverse=True)
    #select the required no of items
    top_similar_items = similar_items[1:top+1]

    #get the indices of the recommended items
    recommended_items = [x[0] for x in top_similar_items]

    def get_first_url(url):
        return url.split('|')[0]

    #return the recommended item details
    item_details = df.iloc[recommended_items][['Name','Brand','Rating']]
    item_details['ImageURL'] = df.iloc[recommended_items]['ImageURL'].apply(get_first_url)


    return item_details
```

This function implements a content-based filtering approach to recommend similar items from the dataset. It utilizes Tf-Idf (Term Frequency- Inverse Document Frequency) and cosine similarity to find the similarity between item tags and recommend the top similar items.

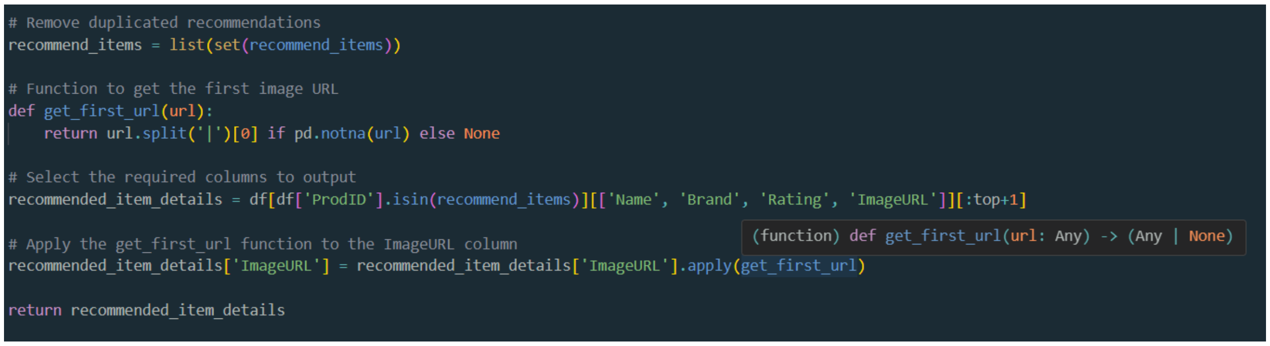
The function takes the dataset, item name and the number of recommendations needed as the parameters. Initially, the function checks if the provided item is present in the dataset. If not, it prints a message that the item is not available. Moreover, Tfidf vectorizer transforms the tags into numerical vectors while removing common English stop words and the cosine similarity between each item is calculated.

Furthermore, function finds the index of the target item in the data frame. Then it enumerates over the cosine similarity matrix and retrieves the similarities of the target item with other items. Items are sorted in descending order of their cosine similarity values and the top similar items are selected. Finally, item details such as name, brand, rating and image URL of the top similar items are returned.

- Collaborative filtering (User-based)



```python
def userbased_recommendation(df, target_user, top):
    # Create user-item matrix where rows are users, columns are product IDs, and the values are ratings
    user_item_matrix = df.pivot_table(index='ID', columns='ProdID', values='Rating', aggfunc='mean').fillna(0).astype(int)

    # Calculate cosine similarity between each user
    user_similarities_matrix = cosine_similarity(user_item_matrix)

    # Get the index of the target user
    target_user_index = user_item_matrix.index.get_loc(target_user)

    # Get the similarity score of the target user with other users
    similarity_of_user = user_similarities_matrix[target_user_index]

    # Sort the similarities in descending order excluding index 0
    similar_user_indices = similarity_of_user.argsort()[::-1][1:]

    # Create a list to add recommended items
    recommend_items = []

    for user_index in similar_user_indices:
        # Get the product ratings of similar users
        rated_by_similar_user = user_item_matrix.iloc[user_index]

        # Find products that similar user has rated but the target user has not
        not_rated_by_target_user = (rated_by_similar_user == 0) & (user_item_matrix.iloc[target_user_index] == 0)

        # Add up to 5 products per similar user
        recommend_items.extend(user_item_matrix.columns[not_rated_by_target_user][:5])

    # Remove duplicated recommendations
    recommend_items = list(set(recommend_items))
```



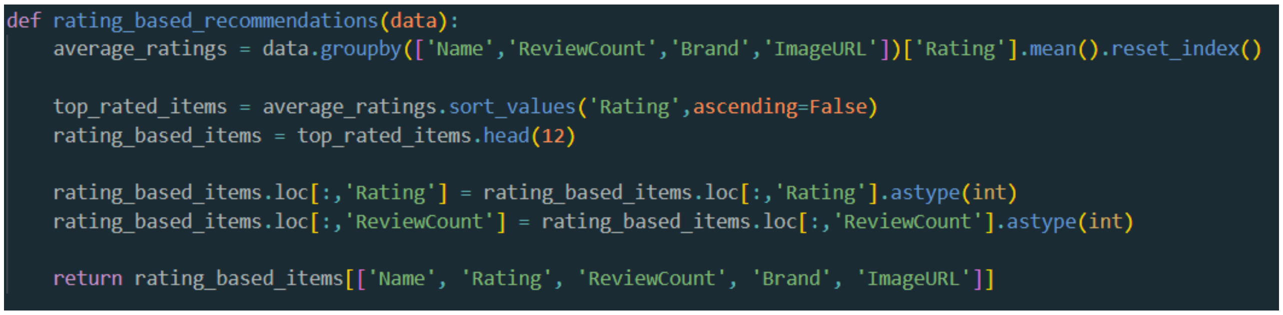
```python
# Remove duplicated recommendations
recommend_items = list(set(recommend_items))

# Function to get the first image URL
def get_first_url(url):
    return url.split('|')[0] if pd.notna(url) else None

# Select the required columns to output
recommended_item_details = df[df['ProdID'].isin(recommend_items)][['Name', 'Brand', 'Rating', 'ImageURL']][:top+1]

# Apply the get_first_url function to the ImageURL column          (function) def get_first_url(url: Any) -> (Any | None)
recommended_item_details['ImageURL'] = recommended_item_details['ImageURL'].apply(get_first_url)

return recommended_item_details
```

This function implements a user-based collaborative filtering approach to recommend products by finding similar users based on their product ratings. The function takes the data frame that contains user ids, product ids and ratings. Additionally, the function requires the target user id and the number of recommendations to return.

The user-item interaction data is transformed into matrix using the pivot_table function in pandas, where rows represent users, columns represent product IDS and values represent ratings. Missing values are filled with 0, indicating that a user has not rated a particular product.

Cosine similarity is used to find the similarities between each user based on their rating vectors. This allows the function to measure how closely a user's preference align with others. Next, the function identifies the target user in the matrix and retrieve their similarity scores with other users. These similarity scores are then sorted in descending order to prioritize the most similar users.
For the recommendation logic, for each similar user, the function checks which products have been rated by that user but not by the target user and 5 such products are taken from each user. Finally, the function collects and combine these recommendations ensuring no duplicate products are suggested and return the top recommendations.

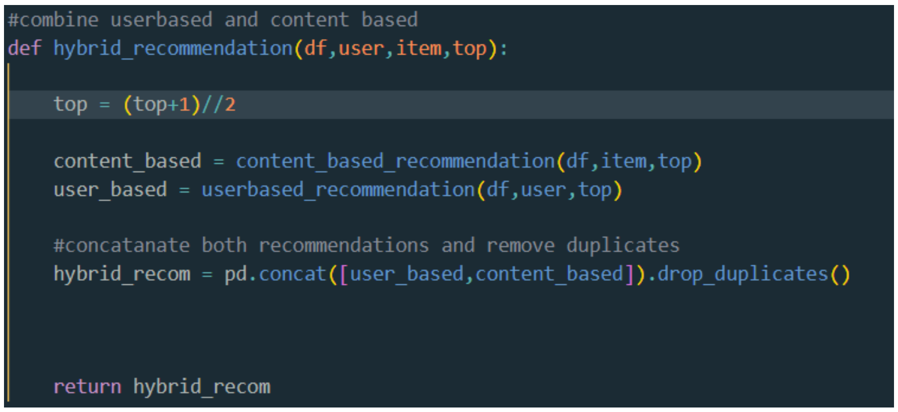
- ## Rating Based Recommendations



```python
def rating_based_recommendations(data):
    average_ratings = data.groupby(['Name','ReviewCount','Brand','ImageURL'])['Rating'].mean().reset_index()

    top_rated_items = average_ratings.sort_values('Rating',ascending=False)
    rating_based_items = top_rated_items.head(12)

    rating_based_items.loc[:,'Rating'] = rating_based_items.loc[:,'Rating'].astype(int)
    rating_based_items.loc[:,'ReviewCount'] = rating_based_items.loc[:,'ReviewCount'].astype(int)

    return rating_based_items[['Name', 'Rating', 'ReviewCount', 'Brand', 'ImageURL']]
```

This function generates a list of top-rated items based on average product ratings making it suitable for recommending high quality products. The function takes the data frame as the input and groups the data by product name, review count, brand and image URL. For each group, it calculates the mean rating. Next, the products are sorted in descending order based on their average ratings. Finally, it extracts the top 12 highly rated items and return as a data frame.

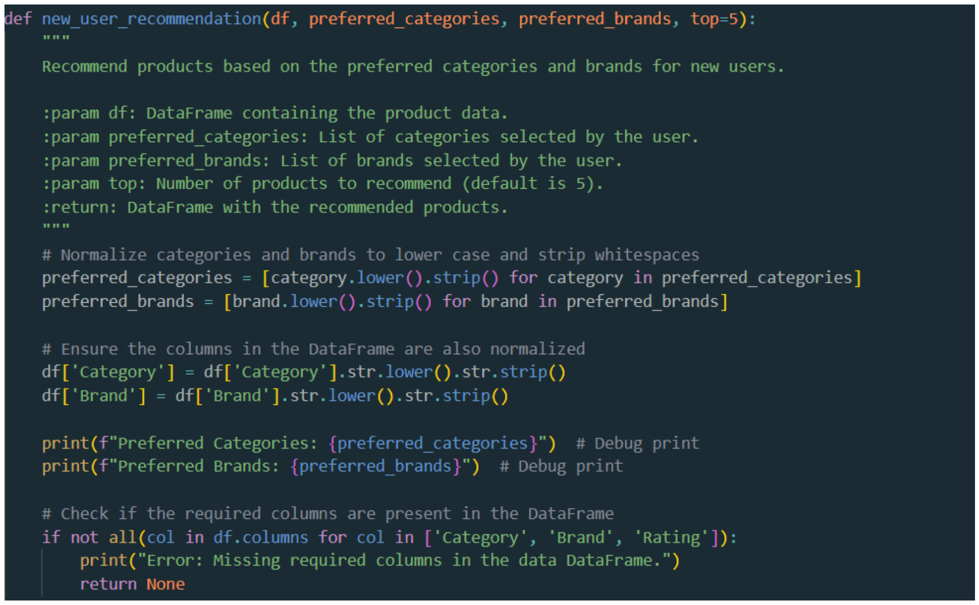
- ## Hybrid approach



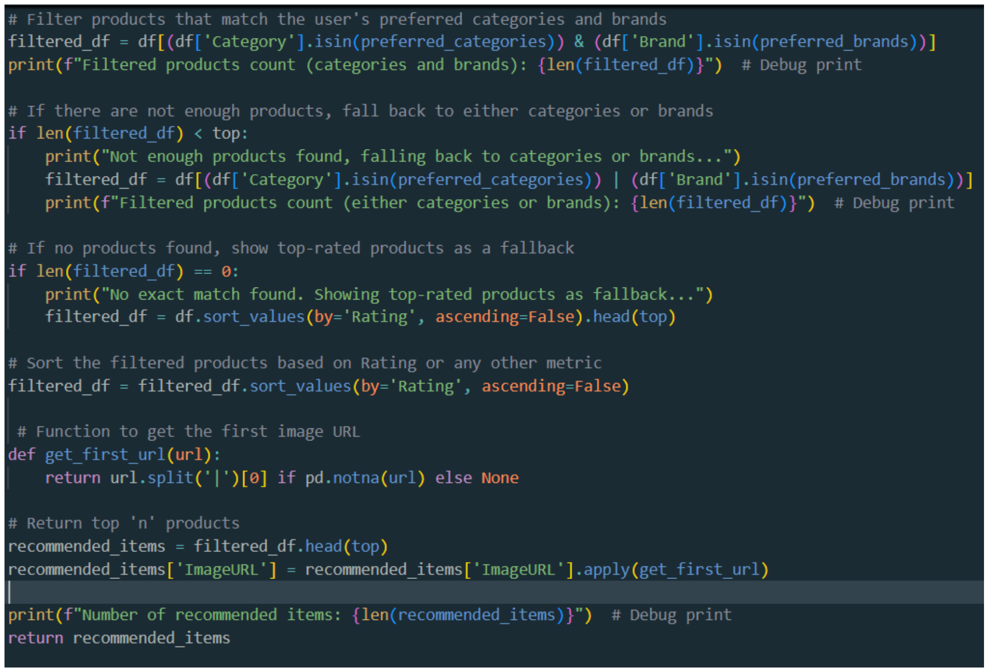
```python
#combine userbased and content based
def hybrid_recommendation(df,user,item,top):

    top = (top+1)//2

    content_based = content_based_recommendation(df,item,top)
    user_based = userbased_recommendation(df,user,top)

    #concatanate both recommendations and remove duplicates
    hybrid_recom = pd.concat([user_based,content_based]).drop_duplicates()



    return hybrid_recom
```

This function combines both content-based and user-based approaches to provide a balanced set of recommendations. Function takes the dataset, user id, item name and number of recommendations needed as the parameters. First, number of requested recommendations is divided by 2 to balance between user based and content-based recommendations. It then calls the content-based and user-based functions and pass the required parameters. Finally, the two sets of recommendations are concatenated, and duplicates are removed to ensure a unique list of recommendations.

- Custom function for recommending products based on user preferences



```python
def new_user_recommendation(df, preferred_categories, preferred_brands, top=5):
    """
    Recommend products based on the preferred categories and brands for new users.

    :param df: DataFrame containing the product data.
    :param preferred_categories: List of categories selected by the user.
    :param preferred_brands: List of brands selected by the user.
    :param top: Number of products to recommend (default is 5).
    :return: DataFrame with the recommended products.
    """
    # Normalize categories and brands to lower case and strip whitespaces
    preferred_categories = [category.lower().strip() for category in preferred_categories]
    preferred_brands = [brand.lower().strip() for brand in preferred_brands]

    # Ensure the columns in the DataFrame are also normalized
    df['Category'] = df['Category'].str.lower().str.strip()
    df['Brand'] = df['Brand'].str.lower().str.strip()

    print(f"Preferred Categories: {preferred_categories}")  # Debug print
    print(f"Preferred Brands: {preferred_brands}")  # Debug print

    # Check if the required columns are present in the DataFrame
    if not all(col in df.columns for col in ['Category', 'Brand', 'Rating']):
        print("Error: Missing required columns in the data DataFrame.")
        return None
```



```python
# Filter products that match the user's preferred categories and brands
filtered_df = df[(df['Category'].isin(preferred_categories)) & (df['Brand'].isin(preferred_brands))]
print(f"Filtered products count (categories and brands): {len(filtered_df)}")  # Debug print

# If there are not enough products, fall back to either categories or brands
if len(filtered_df) < top:
    print("Not enough products found, falling back to categories or brands...")
    filtered_df = df[(df['Category'].isin(preferred_categories)) | (df['Brand'].isin(preferred_brands))]
    print(f"Filtered products count (either categories or brands): {len(filtered_df)}")  # Debug print

# If no products found, show top-rated products as a fallback
if len(filtered_df) == 0:
    print("No exact match found. Showing top-rated products as fallback...")
    filtered_df = df.sort_values(by='Rating', ascending=False).head(top)

# Sort the filtered products based on Rating or any other metric
filtered_df = filtered_df.sort_values(by='Rating', ascending=False)

 # Function to get the first image URL
def get_first_url(url):
    return url.split('|')[0] if pd.notna(url) else None

# Return top 'n' products
recommended_items = filtered_df.head(top)
recommended_items['ImageURL'] = recommended_items['ImageURL'].apply(get_first_url)

print(f"Number of recommended items: {len(recommended_items)}")  # Debug print
return recommended_items
```

This function implements a recommendation logic for new users based on their preferred brands and categories. It first normalizes the user preferences and the product data to ensure consistency, then filter the products that match both the selected categories and brands. If there are not enough exact matches, we used a fallback mechanism where it broadens the filter to allow matches on either the category or brand. Furthermore, the filtered products are then sorted by their ratings, ensuring that the highest rated products are prioritized. Finally, the top n products are recommended to the user.

Evaluation and Testing

- **User Testing**

Three of our colleagues were given our system to test out and provide feedback. These are their comments.

First person: Sandeepa is the first person. He was pleased with the suggestions our system made. He claimed that because our system offers personalized recommendations for users and suggests related products when a user searches for information, it is more efficient.

Second person: Panduka is the second person. Because our system makes both user and content-based recommendations for all user and products, he was happy with it. He claimed that more advanced user interfaces are preferable.

Third person: Rangika is the third person, he is a techy guy. He felt satisfied with our system because it helps new users get started by suggesting trending and user-preferred products. He also mentioned that this would be a fantastic solution for the project because our system offers a wide range of recommendation ways. He suggests adding advanced models based on machine learning to the system to make it even better.

- **Challenges Faced and Solutions**

Data Sparsity:

- Challenge: The given data also lacked complete information about users and their preferences, resulting in the fact that there was little user-product interaction data in the dataset.
- Solution: To counter this, hybrid recommendation techniques were used. This was because the system achieved better results by integrating both the collaborative filtering and content-based filtering methods even when little user interaction data was provided.

Developing Custom recommendation logic for new users:

- Challenge: There was not a specific recommendation logic to address cold start problem.
- Solution: To counter this, we developed a function that retrieves user preferences from the database and recommend them based on the categories and brands they selected during sign up.

Conclusion and Future works

Conclusion: The project develops an ecommerce recommendation system that adopts several machine learning methods to provide personalized product recommendations for users with the aim of maximizing user shopping experiences. We integrated the content-based approach, collaborative filtering techniques, rating based and hybrid recommendation systems to implement a model that can fit various user preferences and characteristics of different products. The recommendation system with a Flask-based web application ensures that the interface of the system is smooth and there is ease in the navigation and recommendation to the users. This system can be used to improve customer satisfaction, enhance user activity, or revenue generated for an e-commerce business.

Key achievements include:

- Cold start problem solved – It is challenging to provide recommendations for new users since the recommendation system lacks their data. To address this, we gather information from newly registered users, including their favored brands and categories. We generated recommendations for new users based on that data, our unique recommendation algorithm, and a combination of rating-based techniques.
- Implementation of content-based filtering to recommend products based on attributes.
- Utilization of collaborative filtering models to predict user-item interactions.
- Development of hybrid recommendation models that combine content-based and collaborative filtering techniques.
- Integration of the recommendation system with a user-friendly Flask web interface, enabling efficient product browsing and recommendation display.
- Development of rating-based recommendations that recommends the top-rated products recommended by users.

Limitations of the Current System:

- **Data Availability and Quality:** The recommendation system is totally dependent on the quality and quantity of data, such as user interaction and product attributes. The sparsity in data or loss of values leads to inaccuracies in recommendations.
- **Scalability:** While the dataset grows, especially in big classes comprising millions of users and products, real-time recommendations may become computationally expensive. Scalability optimization remains one of the primary concerns of large-scale e-commerce applications.

Future Work:

- **Real-Time Recommendations:** Implementing real-time recommendation systems that can adapt to user behavior on the fly could be an important enhancement. Streaming data platforms, such as Kafka, can be integrated to enable real-time learning and updates to recommendations.
- **Incorporating More Complex Models:** This may include investigating RNNs or transformers among other deep learning techniques such as reinforcement learning that can yield much subtler recommendations, capturing both the sequential nature of user behavior and temporal evolution.
- Collect more information of users such as their demographic data, browsing history and seasonal changes and provide recommendations based on them.
- Use **React Js** for frontend development and built a responsive web application.