**HelpmateAI – Project**

ShopAssist AI

Part 1: Introduction

Project Background

In today's digital age, online shopping has become the go-to option for many consumers. However, the overwhelming number of choices and the lack of personalized assistance can make the shopping experience daunting. To address this, we have developed ShopAssist AI, a chatbot that combines the power of large language models and rule-based functions to ensure accurate and reliable information delivery.

Problem Statement:

Option 2: Build Your Own Project (BYOP)

* **Product recommendation systems**: Build a content-based product recommendation system that recommends products to users based on their preferences. You may refer to the ‘ShopAssist AI’ project to craft the problem statement for the project.
* We are using a dataset from Kaggle ("[https://www.kaggle.com/work/collections/14270569?privacy=public"](https://www.kaggle.com/work/collections/14270569?privacy=public%22)) containing mobile phone information (mobile names, prices, ratings, etc.) to build a chatbot that parses the dataset and provides accurate mobile phone recommendations based on user requirements. This chatbot, named ShopAssist AI, will:
* Interact with users,
* Understand their mobile phone requirements, and
* Recommend the most suitable phones from the dataset based on their needs and preferences.

Approach:

1. Import the required libraries and Load the Data set
2. Data Preprocessing and Cleaning
3. Prepare the Data as per our need.
4. System Design
5. Conversation and Information Gathering: The chatbot will utilize language models to understand and generate natural responses. Through a conversational flow, it will ask relevant questions to gather information about the user's requirements.
6. Information Extraction: Once the essential information is collected, rule-based functions come into play, extracting top 3 mobiles that best matches the user's needs.
7. Personalized Recommendation: Leveraging this extracted information, the chatbot engages in further dialogue with the user, efficiently addressing their queries and aiding them in finding the perfect mobile solution.

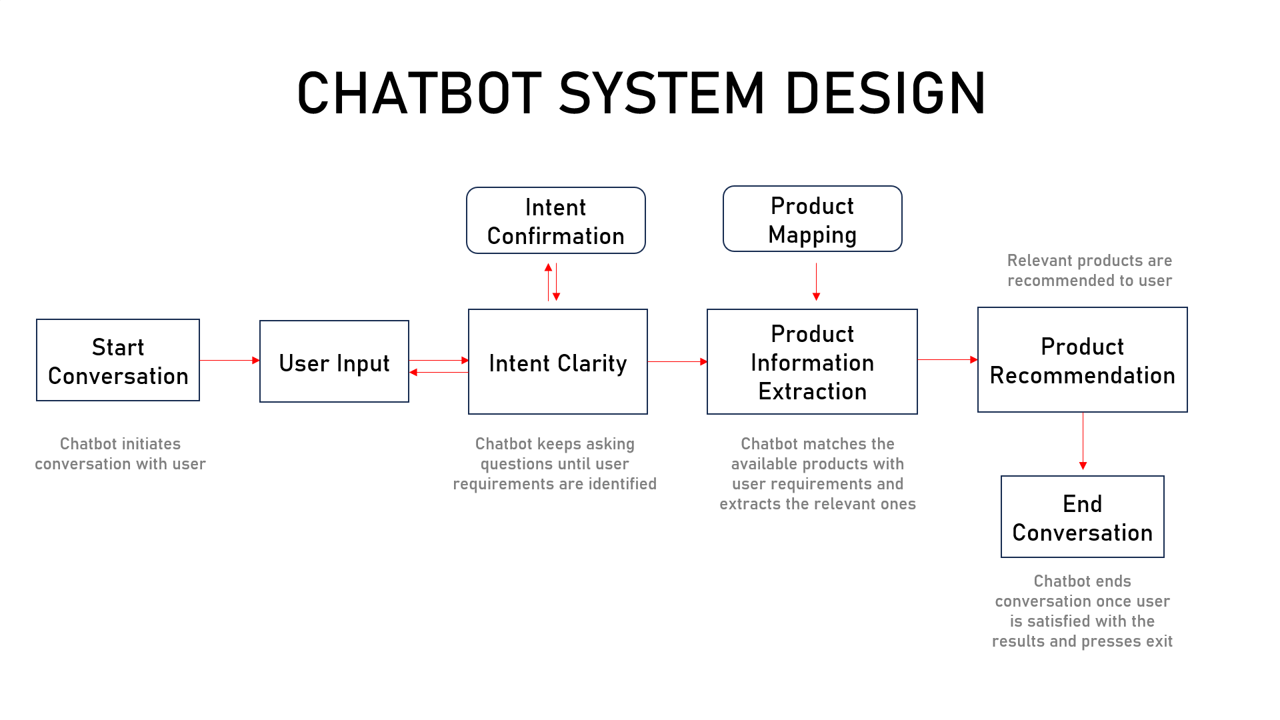
Part 2: System Design

Dataset

We have a dataset mobile\_data.csv where each row describes the features of a single mobile phone. First we will clean and prepare the data for and we also add a one Feature column in the data set at the end. The chatbot that we build will leverage LLMs to parse this Feature column and provide recommendations.

Building the Chatbot

Now let's go ahead and understand the system design for the chatbot.



Stage 1

* Intent Clarity Layer
* Intent Confirmation Layer

Stage 2

* Product Mapping Layer
* Product Information Extraction Layer

Stage 3

* Product Recommendation Layer



Major functions behind the Chatbot

Let's now look at a brief overview of the major functions that form the chatbot. We'll take a deep dive later

* initialize\_conversation(): This initializes the variable conversation with the system message.
* get\_chat\_completions(): This takes the ongoing conversation as the input and returns the response by the assistant.
* moderation\_check(): This checks if the user's or the assistant's message is inappropriate. If any of these is inappropriate, it ends the conversation.
* intent\_confirmation\_layer(): This function takes the assistant's response and evaluates if the chatbot has captured the user's profile clearly. Specifically, this checks if the following properties for the user has been captured or not Ram, Storage, Rating, Display size, Rear Camera, Front Camera, Battery Life and Budget.
* dictionary\_present(): This function checks if the final understanding of user's profile is returned by the chatbot as a python dictionary or not. If there is a dictionary, it extracts the information as a Python dictionary.
* compare\_mobiles\_with\_user(): This function compares the user's profile with the different mobiles and come back with the top 3 recommendations.
* initialize\_conv\_reco(): Initializes the recommendations conversation

Part 3: Implementation

**Note**: Before proceeding, ensure you have your OpenAI API key. This key is necessary for the chatbot to make calls to the OpenAI API. Insert your API key in the designated area in the code.

3.1 : Import the Required Libraries and Load the Data set.

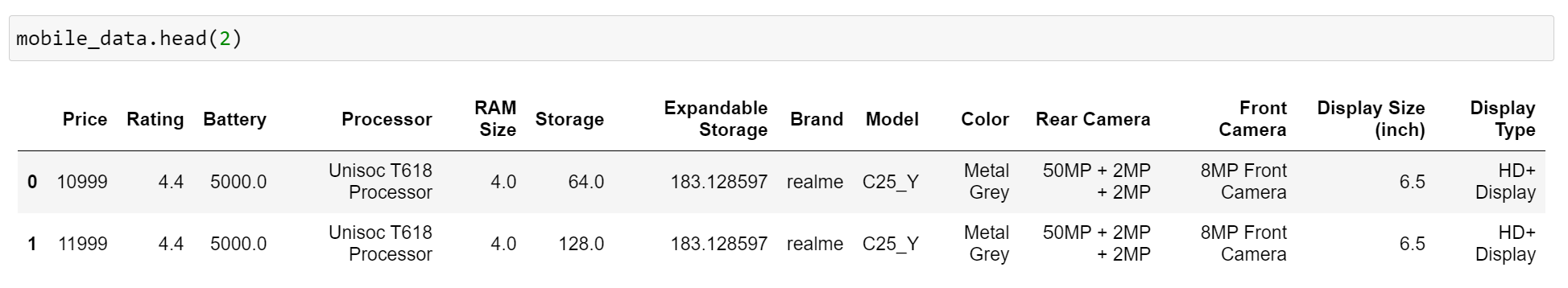


3.2 : Data Preprocessing and Cleaning

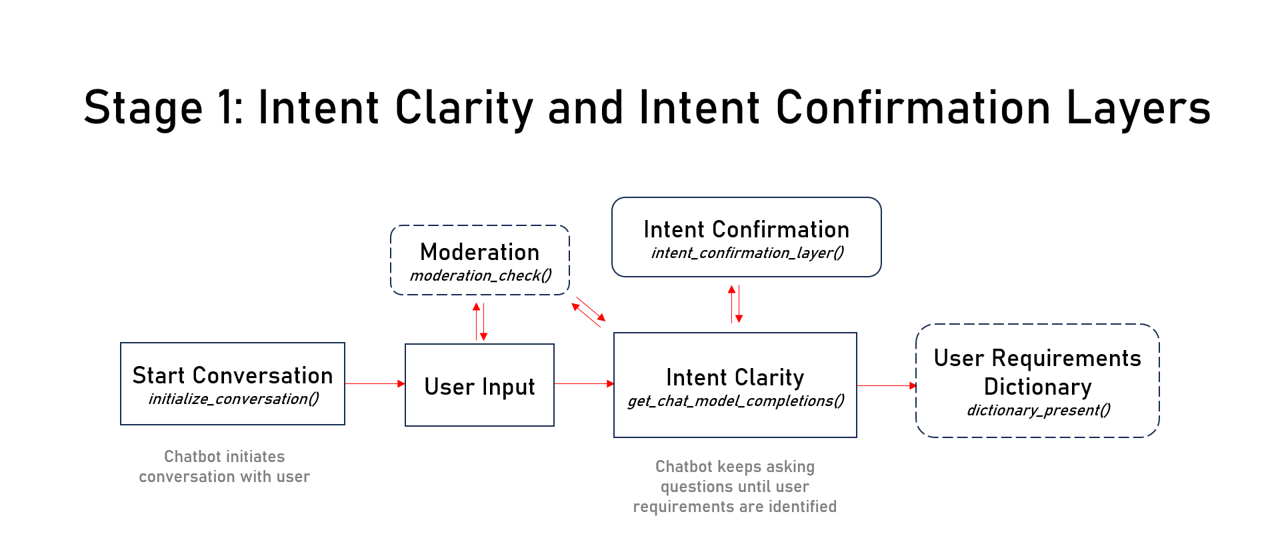
Preparing the data as per our need.

* Remove the rupees symbol from Price column.
* Divide the RAM column into three columns to better understand the Ram size, Internal Storage and Expandable Storage.
* Convert the Mobile\_Name into three different columns Brand, Model and Color of mobile.
* Divide the Camera column into two as Rear Camera and Front camera.
* Extract the battery power from the battery column.
* divide Display column into two as display size in inch and display type.
* Remove unnamed column as it has no use.
* Convert data types of each column in proper manner.
* Check for missing values and remove them.

Now our data set look like,



Stage 1: Intent clarity and Intent Confirmation Layers



Implementing Intent Clarity and Intent Confirmation Layers

Let's start with the first part of the implementation - building the intent clarity and intent confirmation layers. As mentioned earlier, this layer helps in identifying the user requirements and passing it on to the product matching layer. Here are the functions that we would be using for building these layers:

initialize\_conversation():

This initializes the variable conversation with the system message. Using prompt engineering and chain of thought reasoning, the function will enable the chatbot to keep asking questions until the user requirements have been captured in a dictionary. It also includes Few Shot Prompting (sample conversation between the user and assistant) to align the model about user and assistant responses at each step.

``` def initialize\_conversation(): ''' Initializes a conversation for an intelligent mobile phone expert system.

Returns:

list: [{"role": "system", "content": system\_message}]

A list containing a dictionary representing the system's initial message.

The function initiates a conversation with the user, providing guidance on how to interact effectively. The initial message outlines the purpose, instructions for completing a user profile, and examples to demonstrate the interaction.

The system expects the user to provide details related to their mobile phone preferences, such as budget, rating, RAM size, storage, rear camera, front camera, display size (inch), and battery. This information is used to update a dictionary representing the user's preferences, ensuring accurate recommendations.

The conversation flow consists of three main steps:

Understanding User Preferences: Gathering and filling confident values based on user input. Completing the Profile: Using gathered information to fill in remaining preferences. Confirmation: Verifying the updated dictionary to ensure it accurately reflects the user's needs. Users are encouraged to engage meaningfully to ensure their profile is well-represented, allowing for tailored recommendations for the best mobile phone choices.

Usage:

# Initialise the conversation

conversation = initialize\_conversation()

# Print the conversation

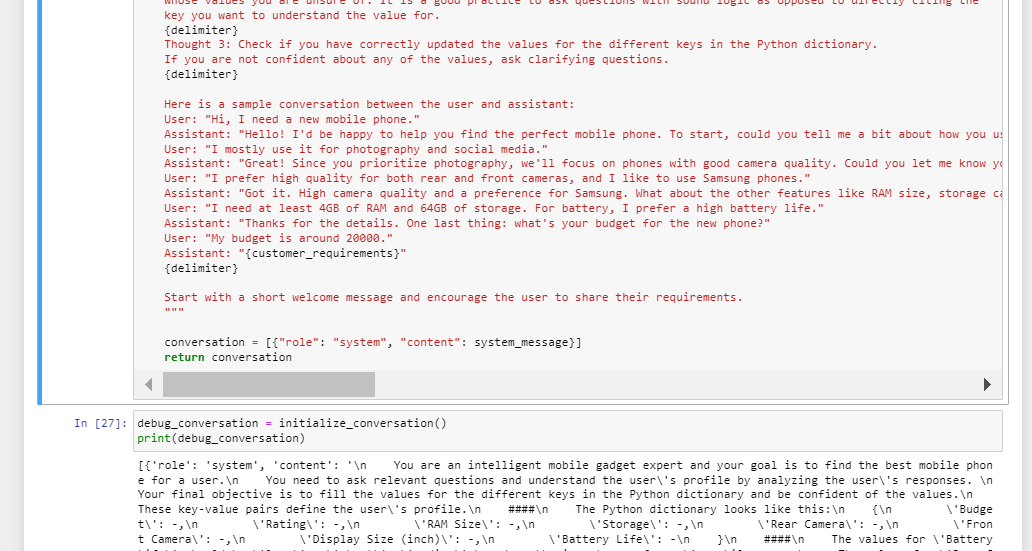
print(conversation)

'''

```

Here are some insights from the jupyter note,





Start with a short welcome message and encourage the user to share their requirements.

Let's now look at the next function.

* get\_chat\_completions(): This takes the ongoing conversation as the input and returns the response by the assistant. We'll use the Chat Completions function for performing LLM calls to OpenAI.

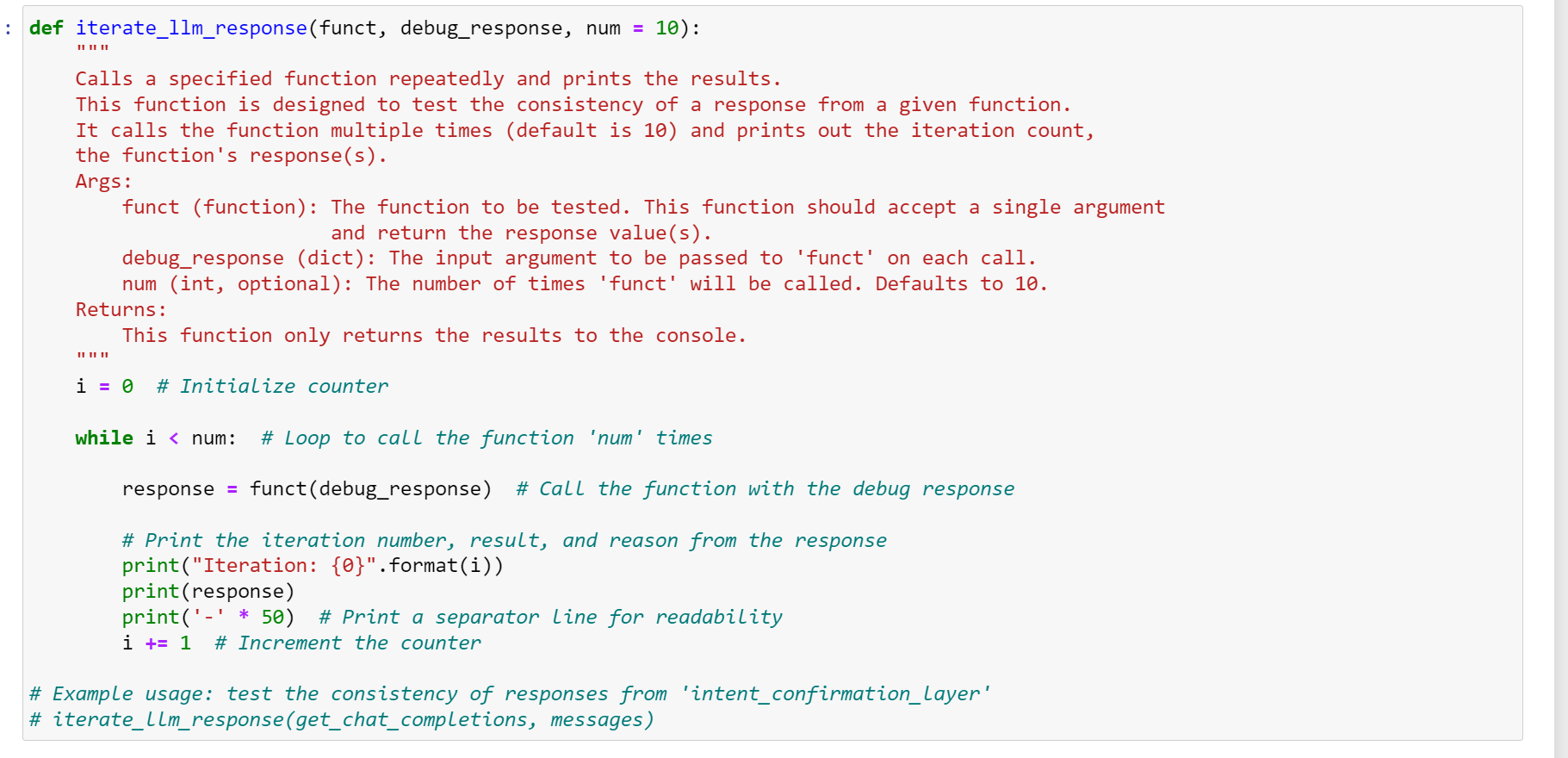
get\_chat\_completions():

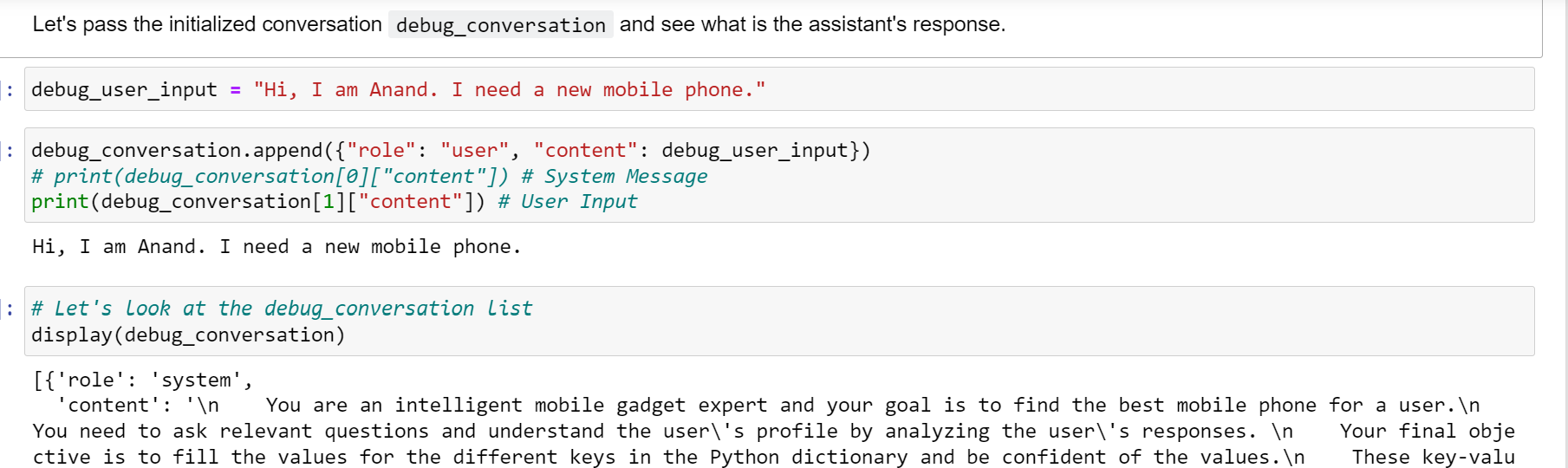
This function perform LLM call using the Chat Completions API to get the LLM response.



iterate\_response() - Helper Function:

We've created a small helper test function to ensure the model's response is consistent. Uncomment the code blocks and run the function iterate\_response(response) to check if the response of the intent\_confirmation\_layer is consistent.

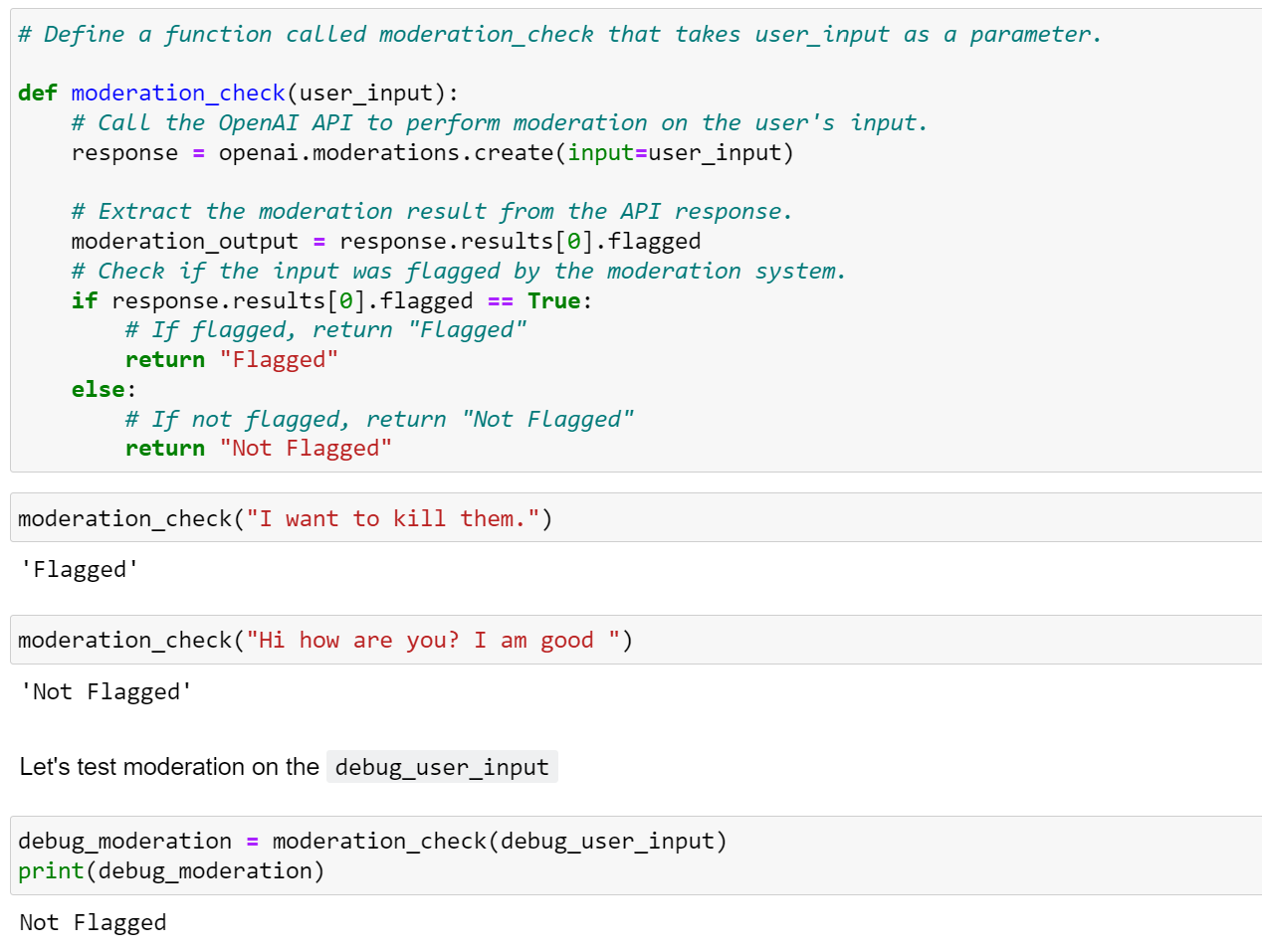




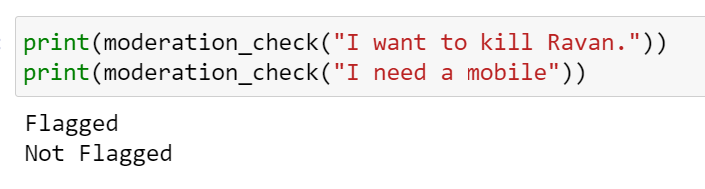
Typically, whenever the chatbot is interacting with the user, all the conversations should be moderated to identify any inappropriate content. Let's look at the function that can help with it.

moderation\_check():

This checks if the user's or the assistant's message is inappropriate. If any of these is inappropriate, you can add a break statement to end the conversation.



Let's now test moderation on some other text.



So, this moderation api may not be perfect but if you ask this to the ChatGPT or it's API (GPT 3.5), it'll not help you with such requests. Remember, moderation should also be applied on the GPT 3.5's output.

Let's now check moderation on the assistant's response debug\_response\_assistant.

As mentioned earlier, you need to understand the user's profile, which essentially means that all the features: Budget, Rating, Ram, Storage, Rear Camera, Front Camera, Display size (inch), and Battery Life are captured or not. Let's look at the function that helps us verify that.

intent\_confirmation\_layer():

This function takes the assistant's response and evaluates if the chatbot has captured the user's profile clearly. Specifically, this checks if the following properties for the user has been captured or not

'Budget'

'Rating'

'RAM Size'

'Storage'

'Rear Camera'

'Front Camera'

'Display Size (inch)'

'Battery Life'

def intent\_confirmation\_layer(response\_assistant):

"""

This function serves as an intent confirmation layer for a smartphone recommendation system.

Parameters:

- response\_assistant (str): The input text containing user requirements captured through 8 keys:

'Rear Camera', 'Front Camera', 'Rating', 'RAM Size', 'Storage', 'Display Size (inch)', 'Battery Life', and 'Budget'.

Returns:

- str: A one-word string in JSON format indicating if the values for the specified keys are correctly filled.

- 'Yes' if the values are correctly filled for all keys ('Rear Camera', 'Front Camera', 'Rating', 'RAM Size',

'Storage', 'Display Size (inch)', 'Battery Life', and 'Budget') based on the importance as stated by the user.

- 'No' otherwise.

Note:

- The values for 'Rear Camera', 'Front Camera', and 'Rating' should be 'high', 'medium', or 'low'.

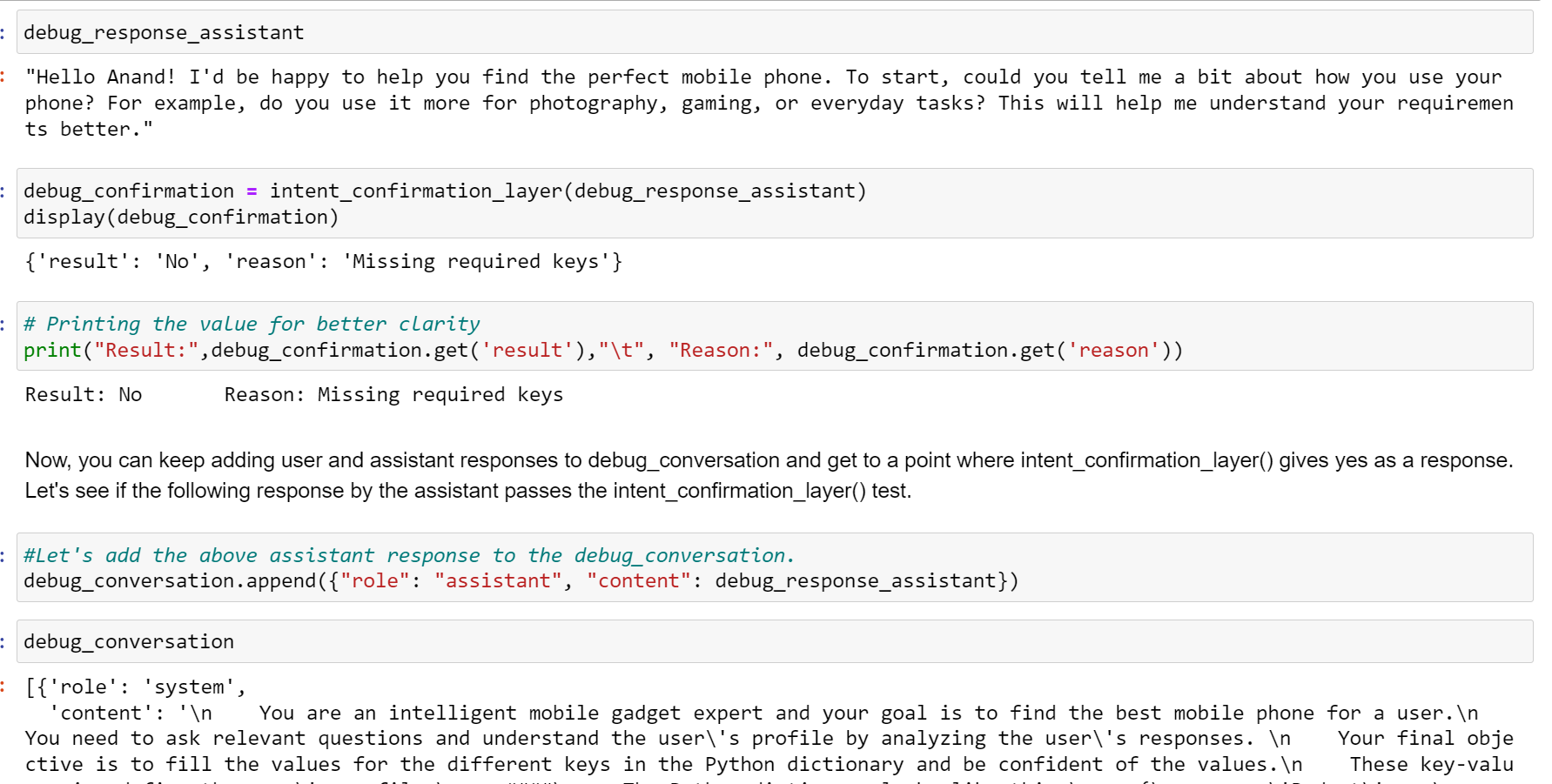
- The value for 'Battery Life' should be 'long', 'medium', or 'short'.

- The other key values ('RAM Size', 'Storage', 'Display Size (inch)', 'Budget') should be in numeric form.

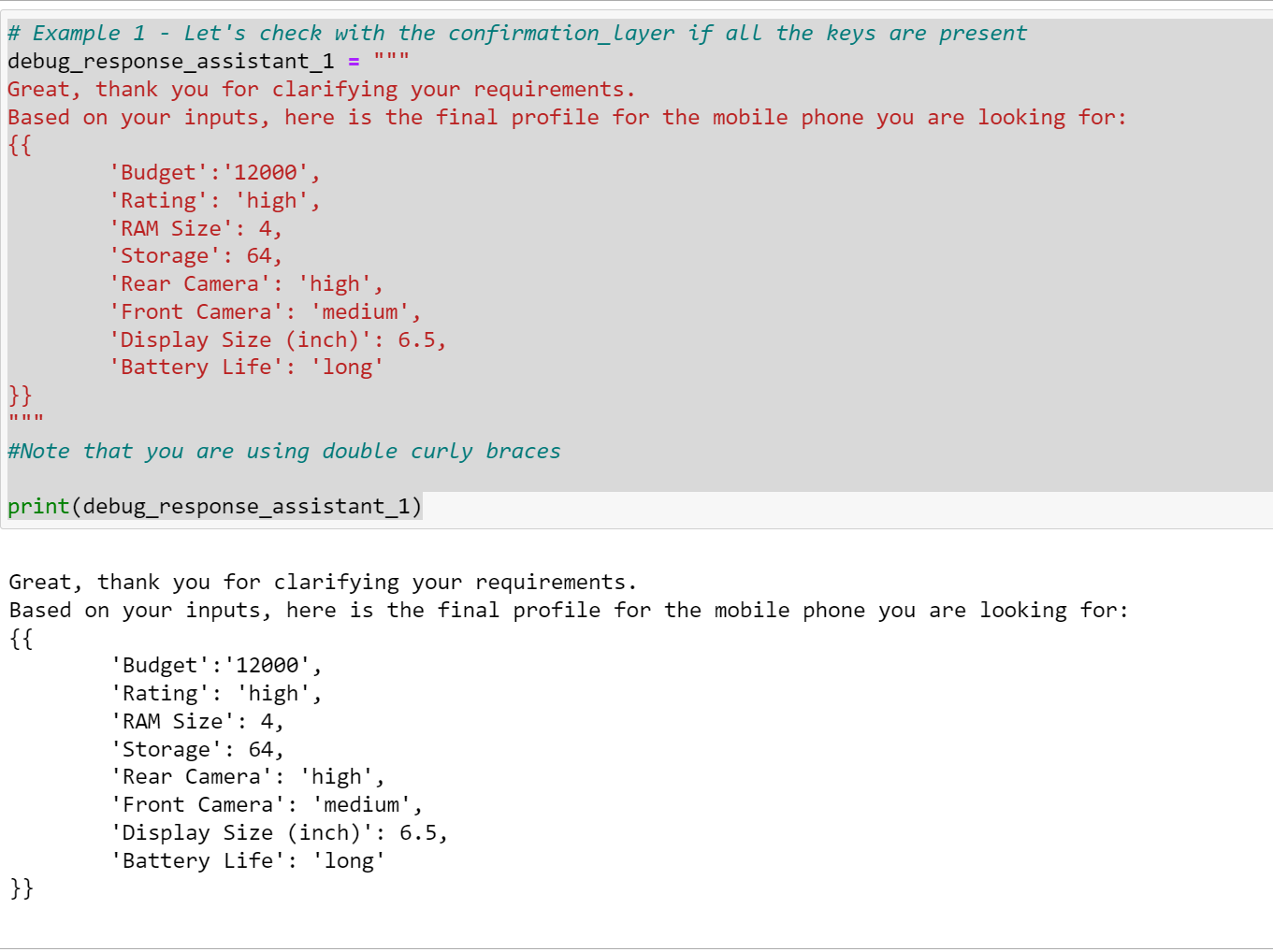
- The input text should be structured such that it contains the necessary keys and their corresponding values.

"""

Let's apply the function to the assistant's reponse and see if it has captured the user profile.

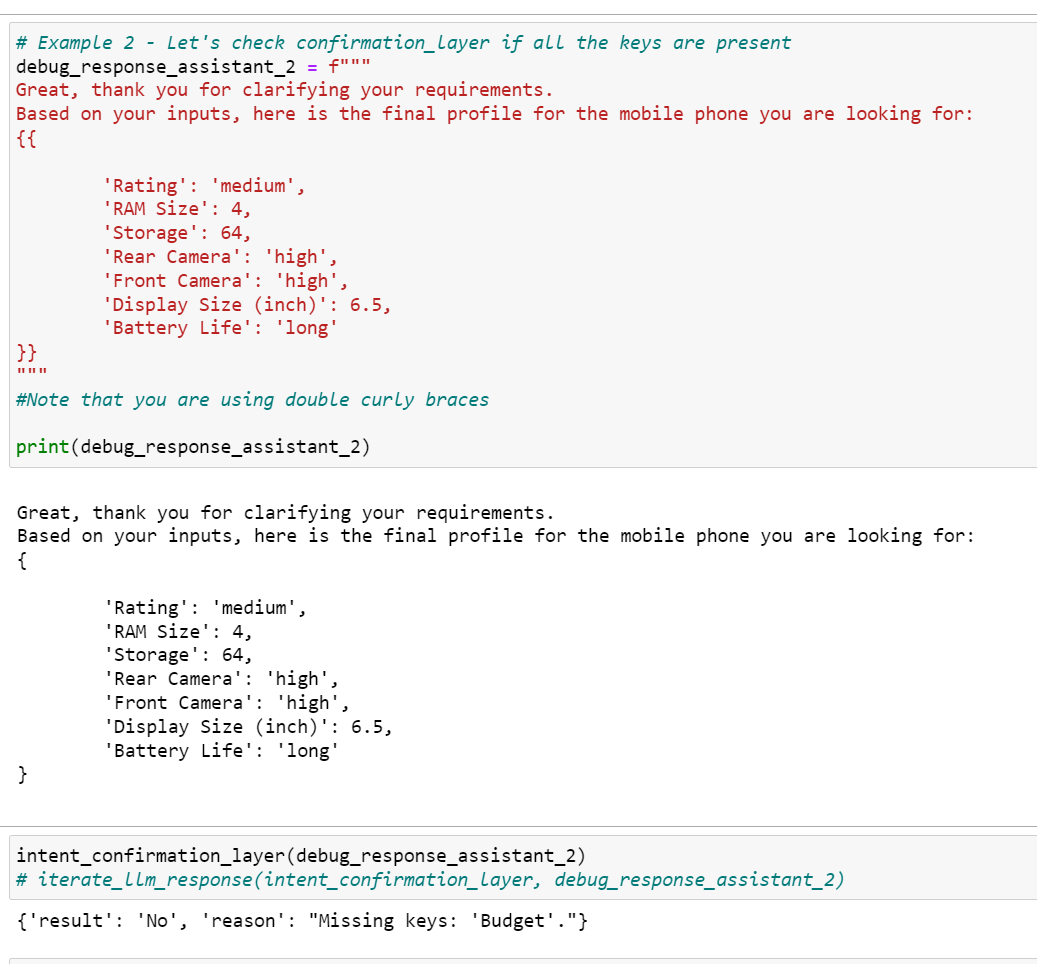


Let's say that after a series of conversations you get the following response from the assistant.



Do you think it'll pass the intent\_confirmation\_layer() test?

Let's try it out.



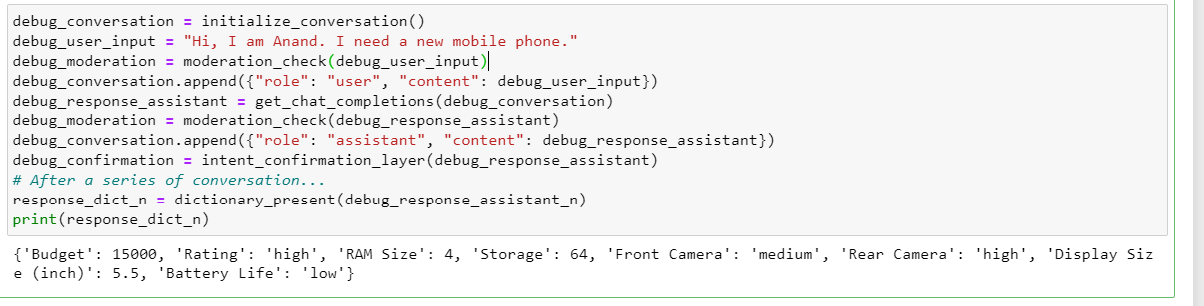
dictionary\_present():

This function checks if the final understanding of user's profile is returned by the chatbot is a Python dictionary or not. This is important as it'll be used later on for finding the right mobiles using dictionary matching.

So we create a dictionary function and check it for LLM function’s consistency.



After iterating LLM”s function we checked our code until we have done.



STAGE 2: Product Map Layer

product\_map\_layer():

This function is responsible for extracting key features and criteria from mobile\_data. Here's a breakdown of how it works:

Use a prompt that assign it the role of a mobile Specifications Classifier, whose objective is to extract key features and classify them based on mobile features.

Provide step-by-step instructions for extracting mobile features from dataset.

Assign specific rules for each feature (e.g., Rear Camera, Front Camera, Rating) and associate them with the appropriate classification value (Low, Medium, or High) and for Battery Life (long, short,medium).

Includes Few Shot Prompting (sample conversation between the user and assistant) to demonstrate the expected result of the feature extraction and classification process.

Create a new column ‘Features’ in the mobile\_data.

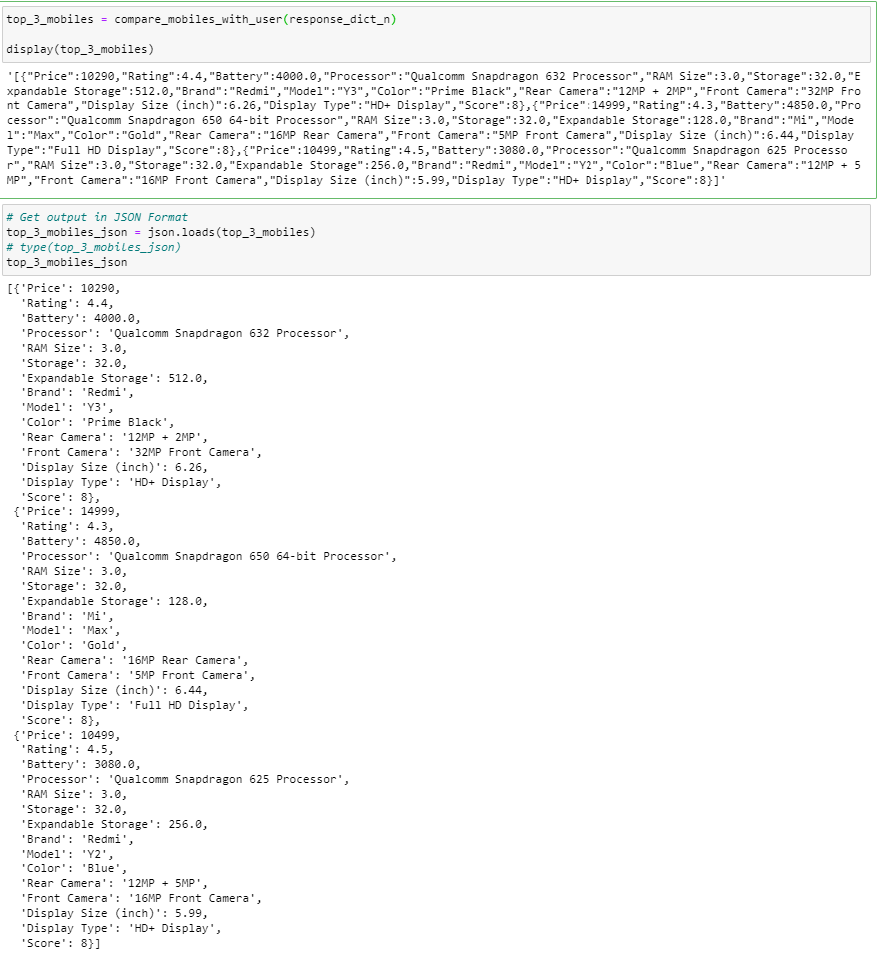
Let’s see one sample mobile feature.



compare\_mobiles\_with\_user():

This function compares the user's profile with the different mobiles and come back with the top recommendations. It will perform the following steps: -

* + It will take the user requirements dictionary as input
  + Filter the mobiles based on their price, keeping only the ones within the user's budget.
  + Calculate a score for each mobile based on how well it matches the user's requirements. - Sort the mobiles based on their scores in descending order.
  + Return the top 3 mobiles as a JSON
  + formatted string.



product\_validation\_layer():

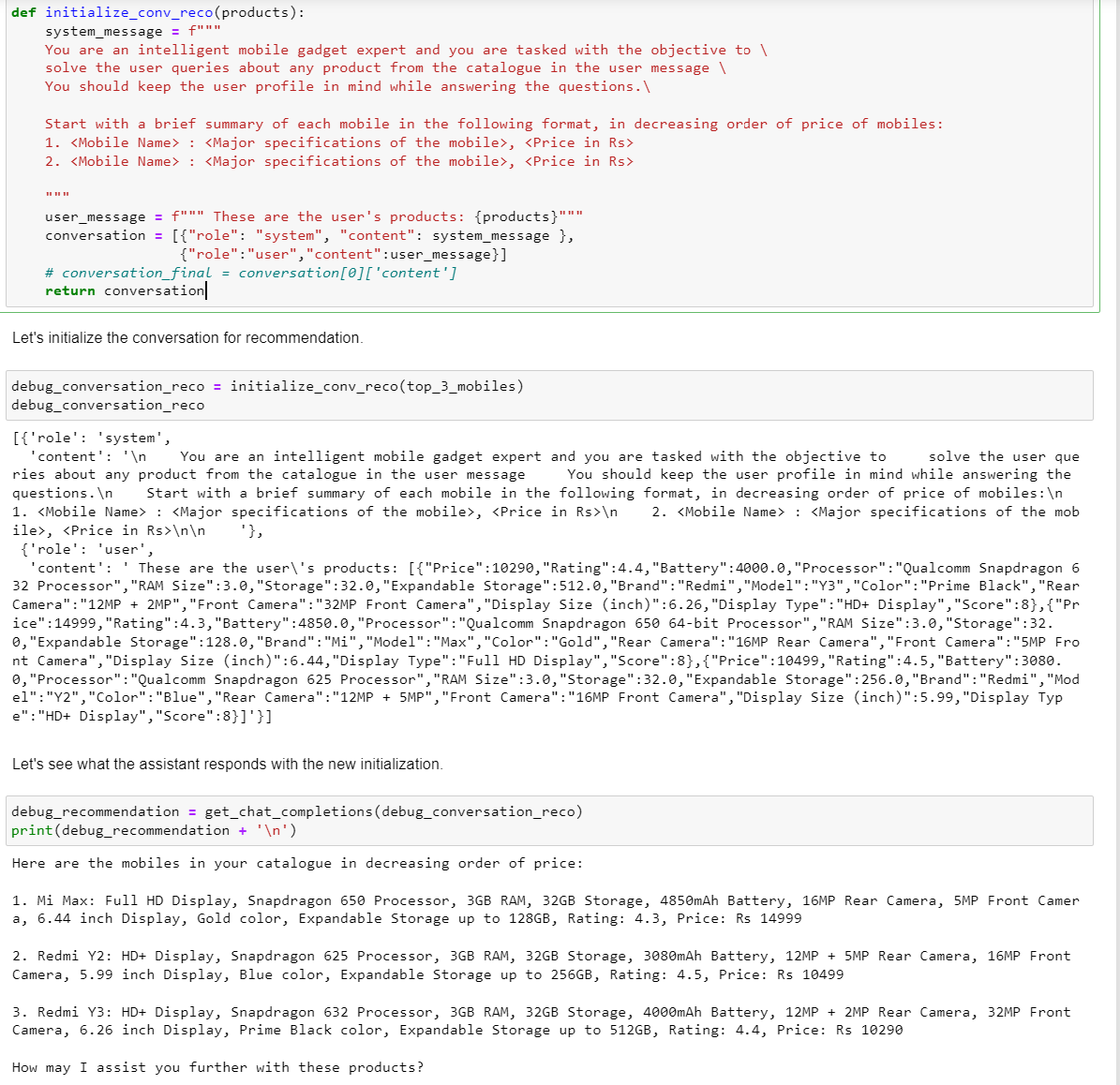
This function verifies that the mobile recommendations are good enough, has score greater than 2, and matches the user's requirements.



STAGE 3: Product Recommendation Layer

Finally, we come to the product recommendation layer. It takes the output from the compare\_mobiles\_with\_user function in the previous layer and provides the recommendations to the user. It has the following steps.

1. Initialize the conversation for recommendation.
2. Generate the recommendations and display in a presentable format.
3. Ask questions basis the recommendations.



STAGE 4: COMBINE ALL 3 STAGE

In this layer, we combine all the three stages that we defined above.

*Stage 1 + Stage 2 + Stage 3*

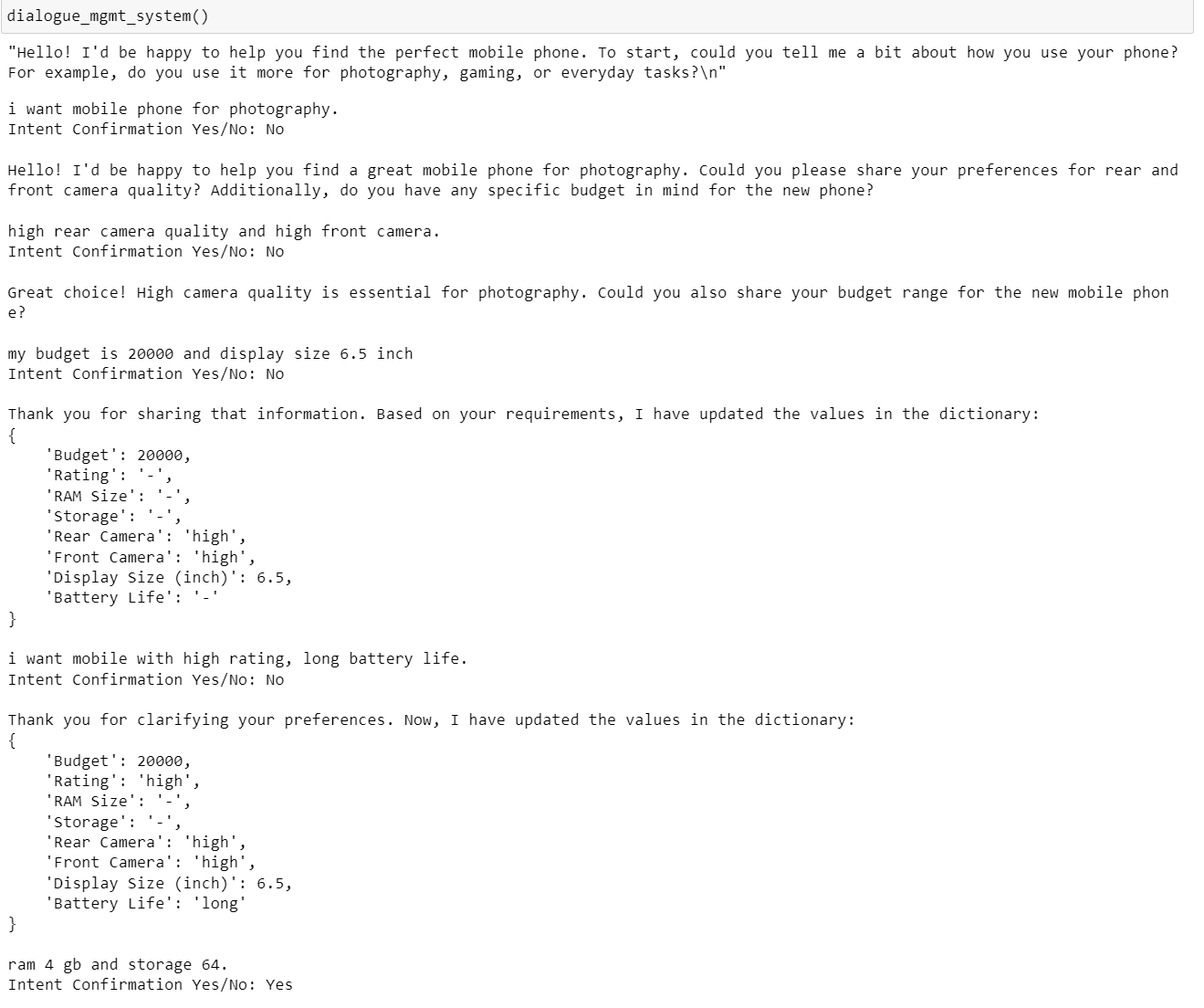
Dialogue Management System

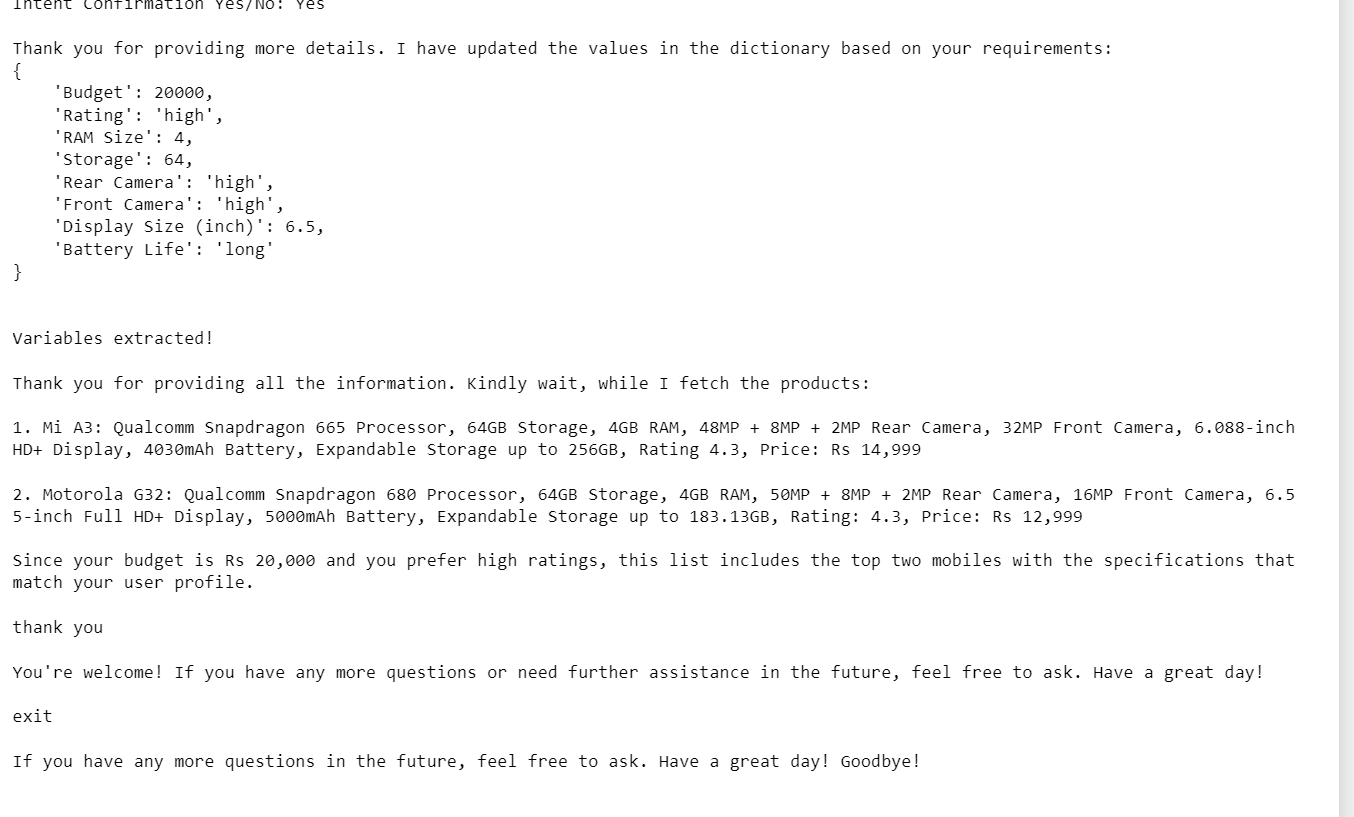
Bringing everything together, we create a diagloue\_mgmt\_system() function that contains the logic of how the different layers would interact with each other. This will be the function that we'll call to initiate the chatbot let’s see some glimps of function,



Finally, we run the final code interact with the Chat Bot.

Let’s see some the screen shot of some interaction with chatbot,





STAGE 5: EVALUATION

Let’s evaluate the code to ensure it meets the requirements and functions as expected.

Let’s see some screenshots of the interaction with the chatbot and evaluate the final implementation.







Conclusion:

In this project, we developed ShopAssist AI, a chatbot designed to assist users in finding the perfect mobile phone based on their preferences and requirements. By leveraging a combination of large language models and rule-based functions, the chatbot effectively interacted with users, gathered their mobile phone requirements, and provided personalized recommendations from a dataset of mobile phones.

Key Achievements:

* Data Preprocessing and Cleaning: Successfully cleaned and prepared the dataset to ensure accurate and relevant information.
* Intent Clarity and Confirmation: Implemented layers to understand and confirm user requirements.
* Product Mapping and Information Extraction: Extracted key features and classified mobile phones based on user requirements.
* Product Recommendation: Developed a system to provide tailored recommendations and engage in further dialogue with users.
* Evaluation: Evaluated the chatbot's performance to ensure it meets the set objectives.

Future Improvements:

* Enhanced NLP Capabilities: Integrate more advanced natural language processing techniques to improve the chatbot's understanding of user inputs.
* Expanded Dataset: Incorporate additional datasets to provide a wider range of mobile phone options and more accurate recommendations.
* User Feedback Loop: Implement a feedback mechanism to continuously improve the chatbot's recommendations based on user feedback.
* Multilingual Support: Add support for multiple languages to cater to a broader audience.
* Integration with E-commerce Platforms: Integrate the chatbot with e-commerce platforms to provide real-time availability and pricing information.

References:

List all references used in project.

* Kaggle. (n.d.). Mobile phone information dataset. Retrieved from https://www.kaggle.com/work/collections/14270569?privacy=public
* OpenAI. (n.d.). API documentation. Retrieved from https://beta.openai.com/docs/
* Python Software Foundation. (n.d.). pandas documentation. Retrieved from https://pandas.pydata.org/docs/
* Scikit-learn developers. (n.d.). Scikit-learn documentation. Retrieved from https://scikit-learn.org/stable/documentation.html
* Hugging Face. (n.d.). Transformers documentation. Retrieved from https://huggingface.co/docs/transformers/index
* Chatbot Design. (n.d.). Best practices for designing conversational interfaces. Retrieved from https://chatbotdesign.io/
* Python Software Foundation. (n.d.). Python documentation. Retrieved from https://docs.python.org/3/