MICHELLE MIRANDA- 22112023

CALORIE TRACKER APP

INTRODUCTION

The Calorie Tracker App is designed to help users track their daily calorie intake and monitor their progress towards their weight goals. It provides a user-friendly interface for entering food and beverage consumption, calculates the total calories consumed, and displays relevant nutritional information. By keeping track of their calorie consumption, users can maintain a balanced diet, achieve weight management goals, and make positive changes to their overall health and well-being.

With the Calorie Tracker App, users can set personalized calorie intake goals based on factors such as age, gender, weight, and activity level. The app then enables them to log their meals and snacks, providing accurate and detailed information about the nutritional content of each item. By having this information readily available, users can make healthier choices, balance their macronutrient intake, and maintain control over their calorie consumption.

USER INTERFACE

A simple command-line interface was implemented that allows users to interact with the calorie tracking app. The user is prompted to either create a new account or login to an existing account. If a new account is to be created, the user is prompted to provide inputs such as name, age, weight, height, activity level, and goal weight. Once logged in, the app presents a menu with several options such as displaying user information, entering food details, displaying calorie consumption information for the day, displaying past daily calorie consumption data, and exiting the app. The user can choose the desired option by entering the corresponding number.

.

DATA STORAGE

The app uses CSV files to store user information and daily calories consumption data of each user. The **'user_info.csv'** file was used to store the user information such as age, height, weight, bmi, daily calorie intake etc. The **'data.csv'** file stores day-wise consumption of total calories, carbohydrates, proteins and fats for each user. CSV is an appropriate way to store user information data due to its simplicity, compatibility and ease of read-write operations. Moreover, the given data is structured with specific columns making CSV a suitable choice to store the data.

A '**food.json'** file was used to store food data such as the calories, carbohydrates, proteins, fats for each food item. JSON uses a simple and easy-to-read format that consists of key-value pairs, making it easy to understand and access the data.

FUNCTIONALITY

The app provides the following basic feature-

1. User Registration – The user creates a new account where the required details such as name, age, weight etc are taken as input and stored in a file.
2. User Login- The user logs in with their unique username which serves as the basis to add or access information from the files.
3. Displays User Information- the user's basic details and some meta operations such as BMI, daily calorie intake and recommended calorie intake are displayed to the user.
4. Food Entry – the user is able to input the type and amount of food consumed in a day.
5. Calorie Calculation and Nutritional Information- The app will automatically calculate the total calories consumed based on the logged food items. The user is provided with their calorie budget, macronutrient intake and the recommended values as well as whether they have exceeded or are lower than their suggested calorie intake. This information is saved and tracked by storing the data in a csv file.
6. Progress Tracking- The user is able to view their past history of daily calorie consumption details

LIBRARIES

The program uses the csv, json and pandas libraries for data storage and manipulation.

IMPLEMENTATION OF PYTHON CONCEPTS-

1. List, Dictionaries
    - Food data such as number of calories, carbohydrates, proteins, fats were stored in a dictionary format in a JSON file
    - MET values with key-value pairs of physical activity and the corresponding MET value was stored in a dictionary
    - Rows from csv files were returned in a list format which was used to access specific values

- Lists were used to store valid input options and check if input is in the list such as for activity level.

2. String & String Functions
    - String formatting was used to display float values such as calories up to 2 decimal places
    - Upper() function was used to ensure input data was in Uppercase
3. Various Operators
    - Addition, subtraction, multiplication and division operations were performed to calculate BMI, total calories consumed, recommended calorie consumption, total calories burned, net calorie consumption etc
4. Conditional Statements
    - Conditional statements were used to validate input responses. Eg: Checking that weight is not greater than 1000.
    - It was used to account for multiple conditions in the BMI interpretation and daily calorie intake functions.
    - It was used to check whether a user exceeded the calorie budget with the help of comparison statements. For example, if net calories was greater than, less than or equal to recommended calories to be consumed to reach goal weight.
    - It was used in the main() function for selecting appropriate functionality based on numeric choice input
5. Looping Statements
    - While loops were used to ensure that the input prompt is provided until the user enters a valid response
    - While loop was used in the main function to display menu options until user chooses to Exit the program
    - For loop was used to iterate over the total items consumed in the day in order to calculate total calorie, carbohydrate, protein and fat consumption in the meal_calorie_calculator(items) function.
6. User Defined Functions
    - Create_account(): The function takes various inputs from the data like name, height, weight etc. and creates an object of UserProfile. The user information is stored in a csv file.

- bmi_interpretation(bmi): This function takes a BMI value as input and returns an interpretation of the BMI category. It uses conditional statements (if, elif, else) to determine the interpretation based on the BMI value. The function checks the BMI against specific ranges and assigns the appropriate interpretation ("Underweight", "Normal weight", "Overweight", or "Obese") accordingly.
- calculate_calories_burned(met_value, weight, duration): This function calculates the calories burned during an activity based on the MET value, weight, and duration. The Metabolic Equivalent of Task (MET) value is a measurement used to estimate the energy expenditure or intensity of physical activities. The calculated calories burned are returned as the output of the function.
- meal_calorie_calculator(items): This function calculates the total calories, fats, carbohydrates, and proteins consumed in a meal based on the food items and their amounts. It takes the number of food items as input (items). It then iterates over each food item, asking the user to enter the food item name and the amount consumed in grams. It retrieves the nutritional information (calories, fats, carbohydrates, proteins) for each food item from the food_items dictionary and calculates the corresponding contribution to the total calories, fats, carbohydrates, and proteins.
- calculate_recommended_macronutrient_intake(total_calories): This function calculates the recommended macronutrient intake (carbohydrates, proteins, fats) based on the total calorie intake. It takes the total calories consumed as input (total_calories) and calculates the calorie distribution for each macronutrient based on percentages (50% carbohydrates, 20% proteins, 30% fats). It then converts the calorie distribution into grams using the caloric values per gram (4 calories per gram for carbohydrates and proteins, 9 calories per gram for fats). The function returns the recommended grams of carbohydrates, proteins, and fats as output.

7. OOPS

- A class called `UserProfile` was created that represents a user in a calorie tracker app. When an instance of the class is created, the `__init__` method initializes the attributes height, weight, age, name etc and calculates the BMI. It then calls the `add_to_csv` method that stores the user profile information in a CSV file.

- The `daily_calories` method calculates the daily calorie requirement for the user based on their gender, weight, height, and activity level. It uses the Mifflin-St. Jeor equation to estimate the Basal Metabolic Rate (BMR) and then adjusts it based on the activity level chosen.

- The `recommended_calorie_consumption` method determines the recommended calorie intake for the user to reach their goal weight. If the user's current weight is greater than the goal weight, it subtracts 500 calories from the daily calorie requirement. If the current weight is lower, it adds 500 calories. If the current weight is equal to the goal weight, it returns the daily calorie requirement as is.

ERROR HANDLING

User inputs are validated to ensure they are within the expected range and format. Try-except blocks are used to catch and handle errors when entering type sensitive data like age. Further conditional statements have been used to help set constraints to ensure input information entered is valid.

For instance, the age input has constraints where the value must lie between 5 and 120. Also, a value error is handled to ensure that only numeric values are received as inputs. Similar validation is performed for other input variables like height, weight, activity level etc.

Try-except blocks are also used to validate whether input response is from specified options such as in the case of activity levels and food items. At every instance the user is informed about the invalid response and a while loop helps ensure that the input prompt is provided until the user enters a valid response.