Project 8 | Building a Calorie intake Calculator

Lawal's Project

2024-11-30

Table of contents

Figure 1: image

0.1 Project Overview

As a Software Engineer in a Health and Leisure company, your task is to add a new feature to the app: a calorie and nutrition calculator. This tool will calculate and display total calories, sugars, fats, and other nutritional values for different foods based on user input.

You have been provided with the `nutrition.json` dataset, which contains the necessary nutritional information for various foods. Each value in the dataset is per **100 grams** of the food item.

0.2 Dataset Summary

Table 1: `nutrition.json`

Column	Description
`food`	The name of the food.
`calories`	The amount of energy provided by the food, measured in kilocalories (kcal) per 100 grams.
`total_fat`	The total fat content in grams per 100 grams.
`protein`	The protein content in grams per 100 grams.
`carbohydrate`	The total carbohydrate content in grams per 100 grams.
`sugars`	The amount of sugars in grams per 100 grams.

0.3 Project instructions

Enhance the Diet Coach app by creating the `nutritional_summary()` function to calculate and return the total nutritional values from the `nutrition_dict` dataset.

Function Output:

- If all the foods are present in the dataset, the function returns a dictionary with keys: `"calories"`, `"total_fat"`, `"protein"`, `"carbohydrate"`, `"sugars"`.
- If any food is missing from the dataset, the function returns the name of the first missing item. Input Format:
- **Dictionary**: For example, calling `nutritional_summary({"Croissants, cheese": 150, "Orange juice, raw": 250}) should output {'calories': 733.5, 'total_fat': 32.0, 'protein': 15.55, 'carbohydrate': 96.5, 'sugars': 38.025}` Here, `150` and `250` represent the grams of each food.
- **Handling non-existent keys**: For example, calling `nutritional_summary({"Croissant": 150, "Orange juice": 250})` should output `"Croissant"`.

0.4 Data Source

The primary data used for this analysis is the nutrition.json, which can be downloaded here. See Table 1 for the column names and descriptions.

0.5 Tools

This project was conducted using `JupyterLab`, a versatile interactive development environment that facilitates data analysis, visualization, and documentation in Python.

0.6 Steps/Explanations

0.6.1 The `json` module was imported, to parse JSON strings or files into Python data structures (like dictionaries and lists) and vice versa.

0.6.2 The `nutrition.json` file was opened in read mode using a context manager, and its contents were parsed into a Python dictionary named `nutrition_dict`.

0.6.3 The function, `nutritional_summary` is designed to compute the cumulative nutritional values (calories, total fat, protein, carbohydrates, and sugars) for multiple food items based on their respective weights in grams. If a food item is not found in `nutrition_dict`, the function identifies it and stops further computation.

```
def nutritional_summary(foods):
```

1. **Input**:

- The function takes a dictionary, `foods`, where:
  - Keys are the names of food items (strings).
  - Values are the weights of those food items in grams (int or float).

2. **Output**:

- **Success Case**: A dictionary (`result_dict`) containing the total nutritional values for all valid food items.
- **Error Case**: If a food item in foods is not found in `nutrition_dict`, the function returns the name of the missing food item (string).

3. **Steps**:

- **Initialization**: A dictionary to store the total accumulated values for each nutritional category is initialized with zeros.

```
result_dict = {
    "calories": 0,
    "total_fat": 0,
    "protein": 0,
    "carbohydrate": 0,
    "sugars": 0
}
```

- **Iteration**: The function iterates over each food item (`name`) and its corresponding weight (`grams`) in the `foods` dictionary.

```
for name, grams in foods.items():
```

- **Check for Existence**:

```
if name in nutrition_dict:
```

- If the food item exists in `nutrition_dict`:
- Nutritional values are calculated using the formula:

$$\text{Value} = \frac{\text{grams} \times \text{nutrition value per 100g}}{100}$$

- These values are added to the corresponding keys in `result_dict`.
- If the food item does not exist: The function immediately returns the name of the missing food item as an error indicator.

```
return name
```

- Final Return: After processing all valid food items, the function returns the accumulated nutritional values.

```
return result_dict
```

0.7 Function Creation

```
import json  # Import the json module to work with JSON files

# Open the nutrition.json file in read mode and load its content into a dictionary
with open('nutrition.json', 'r') as json_file:
    nutrition_dict = json.load(json_file)  # Load the JSON content into a dictionary


def nutritional_summary(foods):
    """
    Calculates the total nutritional values for a given set of foods based on their weights.

    Args:
        foods (dict): A dictionary where keys are food item names (str) and values are their resp

    Returns:
        dict: A dictionary containing the accumulated nutritional values with the following keys:
            - "calories" (float): Total calories.
            - "total_fat" (float): Total fat in grams.
            - "protein" (float): Total protein in grams.
            - "carbohydrate" (float): Total carbohydrates in grams.
            - "sugars" (float): Total sugars in grams.
        str: If a food item is not found in the `nutrition_dict`, the function returns the name o

    Raises:
       KeyError: If the JSON structure in `nutrition_dict` is invalid or lacks necessary keys.
        ValueError: If the input `foods` contains non-numeric values for weights.

    Example:
        >>> foods = {
                "apple": 150,
                "banana": 120,
                "chocolate": 50
            }
        >>> result = nutritional_summary(foods)
        >>> if isinstance(result, str):
                print(f"Missing food item: {result}")
        >>> else:
```

```
            print(result)

    Notes:
        The `nutrition_dict` is loaded from the JSON file `nutrition.json`. It is expected to con
        nutritional information per 100 grams for each food item in the following format:

        {
            "food_name": {
                "calories": value (float),
                "total_fat": value (float),
                "protein": value (float),
                "carbohydrate": value (float),
                "sugars": value (float)
},
            ...
        }
    """

    # Create an empty dictionary with specified keys and initialize each with a value of 0
    result_dict = {
        "calories": 0,
        "total_fat": 0,
        "protein": 0,
        "carbohydrate": 0,
        "sugars": 0
    }
    for name, grams in foods.items():
        if name in nutrition_dict:
            result_dict["calories"] += grams * nutrition_dict[name]["calories"] / 100
            result_dict["total_fat"] += grams * nutrition_dict[name]["total_fat"] / 100
            result_dict["protein"] += grams * nutrition_dict[name]["protein"] / 100
            result_dict["carbohydrate"] += grams * nutrition_dict[name]["carbohydrate"] / 100
            result_dict["sugars"] += grams * nutrition_dict[name]["sugars"] / 100
        else:
            return name  # Return the name of the non-existent food item

    # Return the final accumulated nutritional values
    return result_dict
```

0.8 Limitations

If one item is missing, the function stops and does not compute values for the remaining valid items.

0.9 Recommendations

This function can be enhanced based on the use case. For instance, it can be modified to log missing items, continue processing the remaining valid items, and return both the accumulated nutritional values and a list of missing items.

0.10 References

1. For loop in Intermediate Python Course for Associate Data Scientist in Python Carrer Track in DataCamp Inc by Hugo Bowne-Henderson.
2. Introduction to functions in Python in Intermediate Python Course for Associate Data Scientist in Python Carrer Track in DataCamp Inc by Hugo Bowne-Henderson.
3. Python For Data Analysis 3E (Online) by Wes Mckinney Click here to preview.