HEALTH CALCULATOR INDEX

Email: aditya.25bce10188@vitbhopal.ac.in
REG NO: 25BCE10188

INTRODUCTION

This python programme is built to help user calculate their health index and track their health metrices over time by inputting basic information like weight, height, age, gender and activity level.

The tool calculates three essential values for comparing health

- BMI (body mass index): It calculates body fat based on person's height and weight.
- BMR (Basal Metabolic Rate): the number of calories the body needs at rest.
- TDEE (Total Daily Energy Expenditure): an estimate of daily calorie needs based on lifestyle.

This programme not only displays these metrices but also saves them in a CSV file to build history of the user's health. It generates graphs on current metrics and tracks long-term trends which help users to analyse their health over time.

PROBLEM SOLVING

Tracking personal health metrics such as Body Mass Index (BMI), Basal Metabolic Rate (BMR), and Total Daily Energy Expenditure (TDEE) is essential for individuals who want to monitor their fitness, manage weight, or improve overall well-being. However, many people struggle to calculate these values accurately, record them consistently, and visualize their progress over time. Existing solutions may be complex, require manual logging, or lack long-term tracking capabilities.

To address this, a simple, accessible, and automated system is needed that can:

- Collect basic health-related inputs from the user
- Perform accurate BMI, BMR, and TDEE calculations
- Store each set of results for long-term tracking
- Provide visual feedback through graphs and trends

This project aims to develop a Python-based command-line tool that calculates essential health metrics, stores user data in a history file, and visually presents current and historical values. The system will make it easier for users to understand their health status and monitor changes over time without requiring advanced technical or medical knowledge.

FUNCTIONAL REQUIREMENTS

1. User Input

1.1 The system shall prompt the user to enter their weight (kg).
1.2 The system shall prompt the user to enter their height (meters).
1.3 The system shall prompt the user to enter their age (years).
1.4 The system shall require the user to specify their gender (M/F).
1.5 The system shall display a list of activity level options (Sedentary → Extra Active).
1.6 The system shall allow the user to select an activity level from a 5-option menu.
1.7 The system shall validate inputs and terminate with an error message if invalid.

2. Health Calculations

2.1 The system shall calculate the user’s BMI using weight and height.
2.2 The system shall classify BMI into one of the following categories:

- Underweight
- Normal
- Overweight
- Obese

2.3 The system shall calculate BMR (Basal Metabolic Rate) using the Harris–Benedict equation, adjusted by gender.
2.4 The system shall calculate TDEE (Total Daily Energy Expenditure) using an activity multiplier based on the user’s selected activity level.

3. Result Display

3.1 The system shall display the calculated BMI along with its category.
3.2 The system shall display the calculated BMR (cal./day).
3.3 The system shall display the calculated TDEE (cal./day).

4. History Logging

4.1 The system shall save each entry (weight, height, age, BMI, BMR, TDEE) to a CSV file named health_history.csv.
4.2 If the file does not exist, the system shall create it and include a header row.
4.3 The system shall append new records without overwriting previous ones.

5. Data Loading

5.1 The system shall load existing health history data from health_history.csv.
5.2 The system shall handle the absence of the file by displaying a "No history found" message.

6. Graph Generation

6.1 The system shall display a bar chart showing the current BMI, BMR, and TDEE values.
6.2 The system shall generate a line graph displaying:

- BMI history over time
- TDEE history over time

6.3 The system shall visually distinguish lines using colours, markers, and labels.
6.4 The system shall use grid lines and titles for improved readability.

7. Program Flow

7.1 The system shall begin by collecting user input.
7.2 The system shall perform all calculations based on provided data.
7.3 The system shall store results in the history file.
7.4 The system shall display graphs to the user.
7.5 The system shall end after showing graphs and saving data

NON UNCTIONAL REQUIREMENTS

1. Performance Requirements

1.1 The system shall perform BMI, BMR, and TDEE calculations within **1 second** after the user inputs data.
1.2 The system shall load and append data to the CSV history file without noticeable delay.
1.3 Graphs shall render within **2–3 seconds** depending on system performance.

2. Usability Requirements

2.1 The system shall provide a clear, easy-to-understand command-line interface (CLI).
2.2 All prompts and messages shall be written in simple, user-friendly language.
2.3 The system shall display errors in a readable format and stop execution gracefully.
2.4 Graphs shall contain titles, labels, and legends for clarity.

3. Reliability Requirements

3.1 The system will handle invalid input by terminating with an error message rather than producing incorrect results.
3.2 The system will create the history file automatically if it does not exist.
3.3 The system will append data safely to avoid corrupting the CSV file.
3.4 The system shall accurately read previously stored data without loss.

4. Maintainability Requirements

4.1 The code shall be organized into modular functions to ease debugging and updates.
4.2 Variable and function names shall follow meaningful naming conventions.
4.3 The code shall allow future enhancements such as additional metrics or GUI integration.

5. Portability Requirements

5.1 The system shall run on any device that supports Python 3 (Windows, MacOS, Linux).
5.2 The system shall rely only on standard libraries except for Matplotlib, which can be installed via pip.

6. Security Requirements

6.1 The system shall restrict file operations to the local CSV file and shall not modify other system files.
6.2 User data shall not be transmitted over a network (local-only operation).
6.3 The program shall not store sensitive information beyond health metrics.

SYSTEM REQUIREMENTS

System requirements specify the hardware, software, and environment needed for your program to run successfully.

1. Hardware Requirements

Minimum Requirements

- Processor: 1 GHz or faster
- RAM: 2 GB
- Storage: At least 50 MB free space (to store Python, libraries, and CSV history file)
- Display: Any screen capable of showing CLI text and Matplotlib graphs

Recommended Requirements

- Processor: 2+ GHz multi-core
- RAM: 4 GB or more
- Storage: 100 MB or more
- Display: 1280×720 or higher (for clearer graph output)

2. Software Requirements

Required Software

- Python 3.8 or higher
- Operating system:
  - Windows 10 or later
  - macOS 10.14 or later
  - Linux (any modern distro)

Required Python Libraries

- Matplotlib (for graph plotting)

- csv (standard library)
- os (standard library)
- sys (standard library)

Command to install Matplotlib (if needed):

```
pip install matplotlib
```

3. Operating Environment Requirements

3.1 The program shall run in a command-line/terminal environment (CMD, PowerShell, Bash, etc.).
3.2 The program shall have access to the current working directory to read/write health_history.csv.
3.3 The environment must support pop-up windows for Matplotlib graphs.
3.4 File permissions must allow creating and writing to CSV files.

4. User Requirements

.1 The user must have basic familiarity with entering data via keyboard.
4.2 The user must have permission to install required Python libraries.
4.3 The user must be able to view graphs (GUI display required; not suitable for headless/no-GUI servers unless modified).

5. External Interface Requirements

5.1 The system uses a CLI interface for user input and output.
5.2 The system creates a CSV file for persistent storage.

5.3 The system uses graphical windows (Matplotlib) to display bar charts and line graphs.

DESIGN DIAGRAM

```
│   User Input      │
│ weight, height, │
│ age, gender, │
│ activity level) │


        │
        ▼


│  Calculations     │
│  calculate BMI    │
│  classify BMI     │
│  calculate BMR    │
│  calculate TDEE   │
        │
        ▼
```

```
│       Results        │
│ (BMI, BMR, TDEE │

          │
          ▼

│  Save to CSV History   │
│ save to history () │

        │
        ▼

│    Load History │
│   load history () │

             │
┌───────────┴───────────┐
▼                        ▼
```

```
│ show graphs       │ show history_
│ BMI/BMR/TDEE      │  graph ()
```

DESIGN DECISIONS & RATIONALE

Modular Functions – Each calculation and input/output task is in a separate function for readability, maintainability, and easier testing.

CLI Interface – Simple, lightweight, and cross-platform; no GUI dependencies.

CSV History Storage – Easy, human-readable, and compatible with spreadsheets; avoids database complexity.

Matplotlib Graphing – Provides visual feedback for BMI, BMR, TDEE, and historical trends.

Standard Formulas – Uses WHO BMI and Mifflin-St Jeor BMR formulas for widely accepted accuracy.

Error Handling – Catches invalid inputs and missing files to prevent crashes.

Separation of Concerns – Input, logic, storage, and output are separate for clarity and future extensibility.

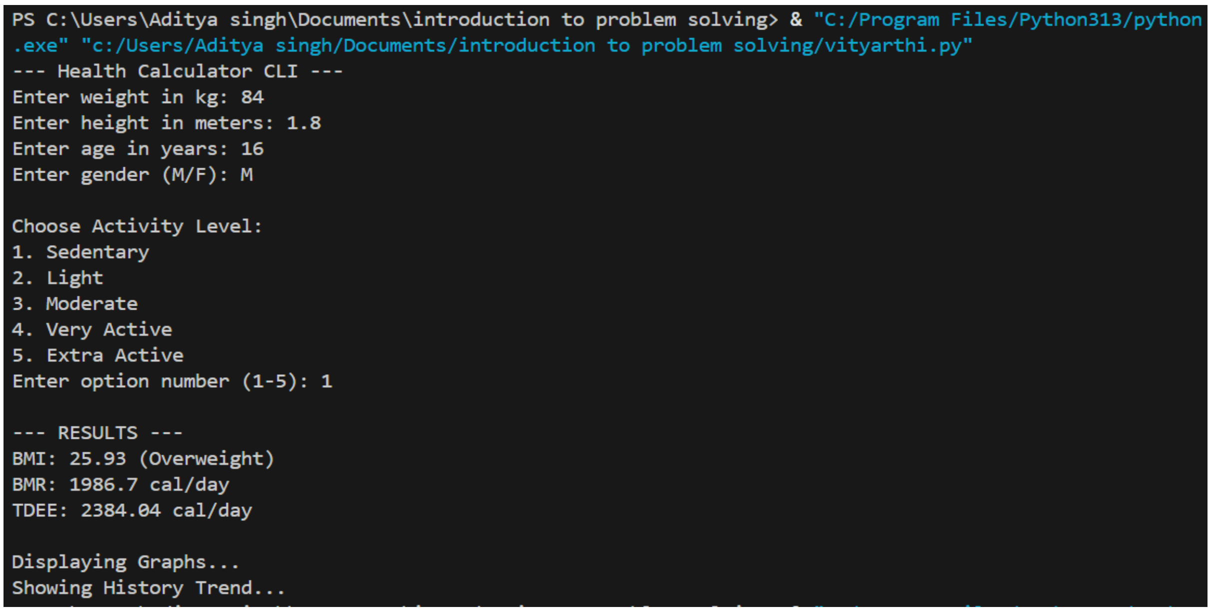
Global File Constant – Easy to configure and maintain the history file path.

Automatic Graph Display – Immediately shows metrics and history to engage users and track progress.

Main Function Orchestration – main () controls program flow, supporting readability and potential expansion.

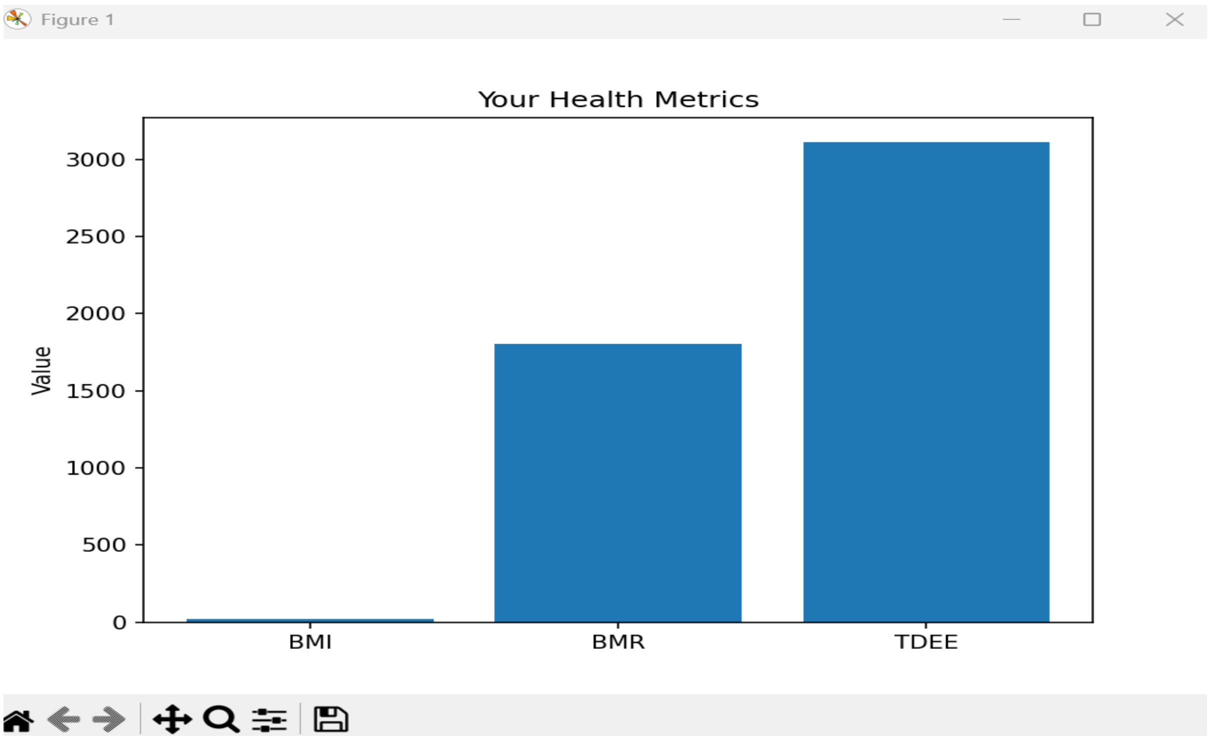
SCREENSHOTS / RESULTS

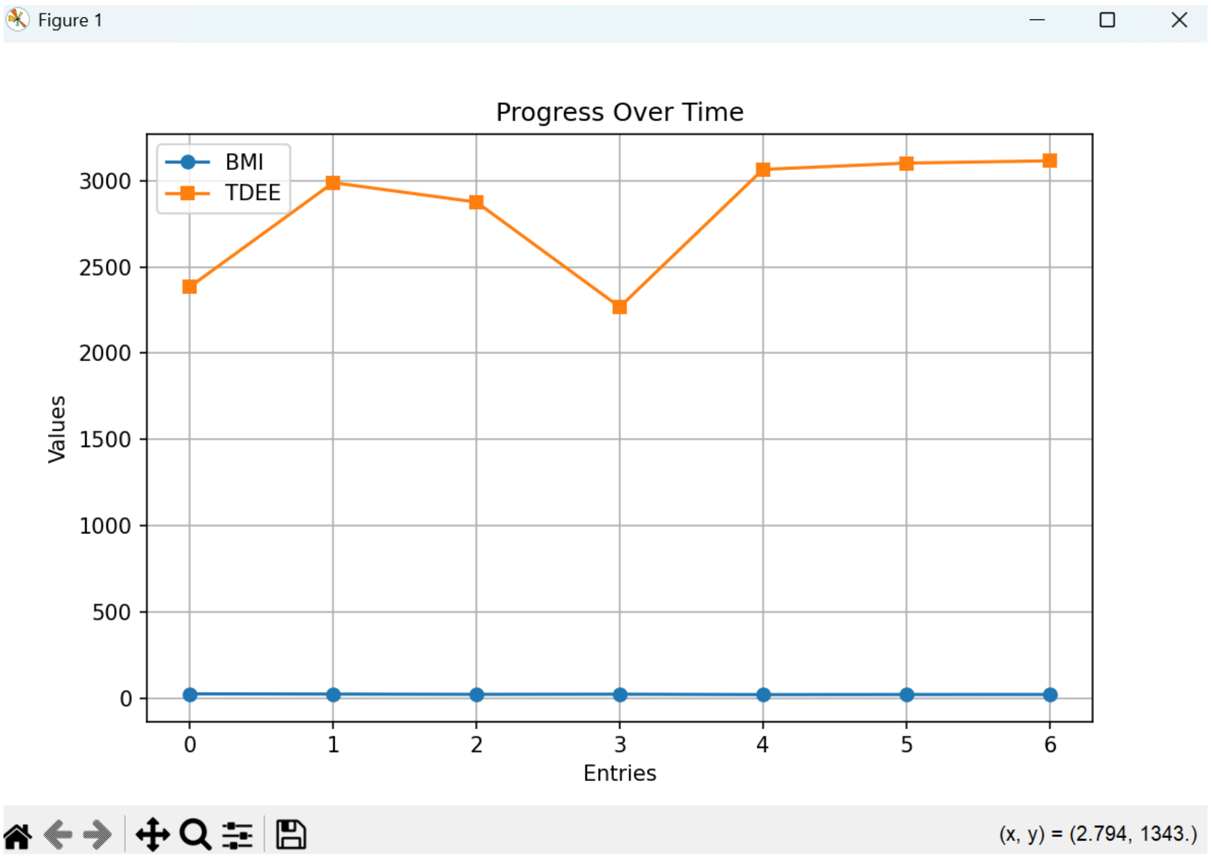
OUTPUT



```
PS C:\Users\Aditya singh\Documents\introduction to problem solving> & "C:/Program Files/Python313/python
.exe" "c:/Users/Aditya singh/Documents/introduction to problem solving/vityarthi.py"
--- Health Calculator CLI ---
Enter weight in kg: 84
Enter height in meters: 1.8
Enter age in years: 16
Enter gender (M/F): M

Choose Activity Level:
1. Sedentary
2. Light
3. Moderate
4. Very Active
5. Extra Active
Enter option number (1-5): 1

--- RESULTS ---
BMI: 25.93 (Overweight)
BMR: 1986.7 cal/day
TDEE: 2384.04 cal/day

Displaying Graphs...
Showing History Trend...
```

GRAPHS





TESTING APPROACH

1. Input Validation Testing

- Test invalid weight, height, age, and gender inputs
- Test invalid activity level selection
- Verify program exits or prompts gracefully without crashing

2. Boundary Testing

- BMI exactly 18.5, 25, 30 → check category assignment
- Age = 0, extremely high age
- Weight/height = 0 or negative

3. Integration Testing

- Full program flow: input → calculation → save → display graphs → load history
- Ensure history CSV is updated correctly and graphs reflect new entries

4. Regression Testing

- After any changes, verify that:
    - BMI/BMR/TDEE calculations remain accurate
    - History logging works correctly
    - Graph plotting still functions

5. Manual/Exploratory Testing

- Test multiple consecutive runs to check history growth
- Visual check of graphs for correctness

6. Optional Automation

- Use pytest for function unit tests
- Mock input() for CLI inputs in automated tests
- Use unittest.mock to avoid actually plotting graphs during automated tests

CHALLENGES IN THE PROGRAMME

Getting User Input Right – Making sure the program doesn't crash when someone enters weird or wrong values, like negative weight or letters instead of numbers.

Handling the History File – Ensuring the CSV file is created if missing, updated correctly, and read without errors.

Avoiding Crashes – Preventing the program from breaking if something unexpected happens, like missing files or invalid data.

Making Graphs Work Smoothly – Getting the charts to display properly on different computers and environments.

Accurate Calculations – Implementing BMI, BMR, and TDEE formulas correctly, especially for tricky edge cases.

Keeping It User-Friendly – Making the prompts and outputs clear so anyone can use it without confusion.

Managing History Data – Ensuring the records grow consistently without breaking the program or creating messy data.

FUTURE ENHANCEMENT

Add a Simple App Interface – Instead of just a command line, a small app with buttons and input fields would make it easier for anyone to use.

Track More Health Metrics – We could include things like body fat percentage, ideal weight, daily water intake, or nutrition suggestions.

Give Personalized Tips – Based on your BMI, TDEE, and activity level, the program could offer diet or exercise recommendations.

Better History & Progress Tracking – Saving data with timestamps and showing weekly or monthly trends would help users see their progress clearly.

Make It Fun – Adding achievements, streaks, or badges to keep users motivated and engaged

REFERENCES

BMI (Body Mass Index) Formula

BMR (Basal Metabolic Rate)

TDEE (Total Daily Energy Expenditure)

Python Standard Library Documentation

- csv module: https://docs.python.org/3/library/csv.html
- os module: https://docs.python.org/3/library/os.html
- sys module: https://docs.python.org/3/library/sys.html

Matplotlib (Graphing Library)

General Python Programming Concepts