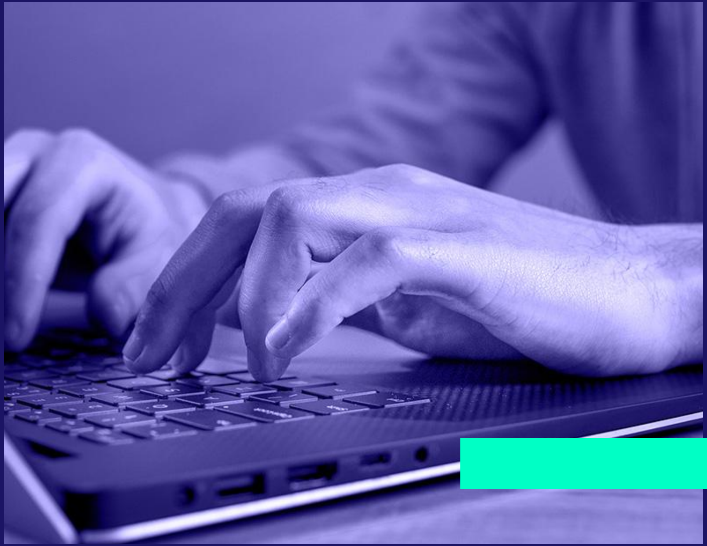
PYTHON MINI PROJECT

BLOOD REPORT ANALYSER

BY KRISH THUKRAL
23FE10CSE00679

INTRODUCTION

This project uses Python to build a blood report analysis tool that evaluates blood report values of the users, offering insights and recommendations to users based on clinical ideal ranges.



OBJECTIVE

★ Create a user-friendly software application for analyzing blood report values.
★ Provide instant feedback on health parameters with actionable suggestions for improvement.
★ Ensure data persistence for future reference and trend tracking.
★ Empower users to monitor their health independently with reliable insights.

LIBRARIES USED

Tkinter

Used to build interactive GUI for the application.
Provides labels, entry boxes, and buttons for user inputs and operations.

Logging

Tracks errors for debugging and monitoring purposes.

Saves logs to a file for further analysis of program behavior.

Messagebox

Displays pop-up messages to notify users about analysis results

JSON

JSON contains key value data pairs

WORKFLOW

User Input

Users enter their blood report values into the GUI for health parameters such as, fasting glucose, and cholesterol.

Validation

The application validates inputs to ensure they are numeric and not left empty.

Analysis

Compares user inputs against ideal ranges stored in a dictionary.
Classifies each parameter as either "Healthy" or "Unideal."

Suggestion

Provides personalized health improvement suggestions for any unideal values based

Results display

Results are displayed in a pop-up message box, with detailed feedback on each health parameter.

Data Persistence

Users can save their health data to a JSON file or load previously saved data for trend analysis and tracking.

FEATURES

JSON

Data persistence using JSON for future reference.

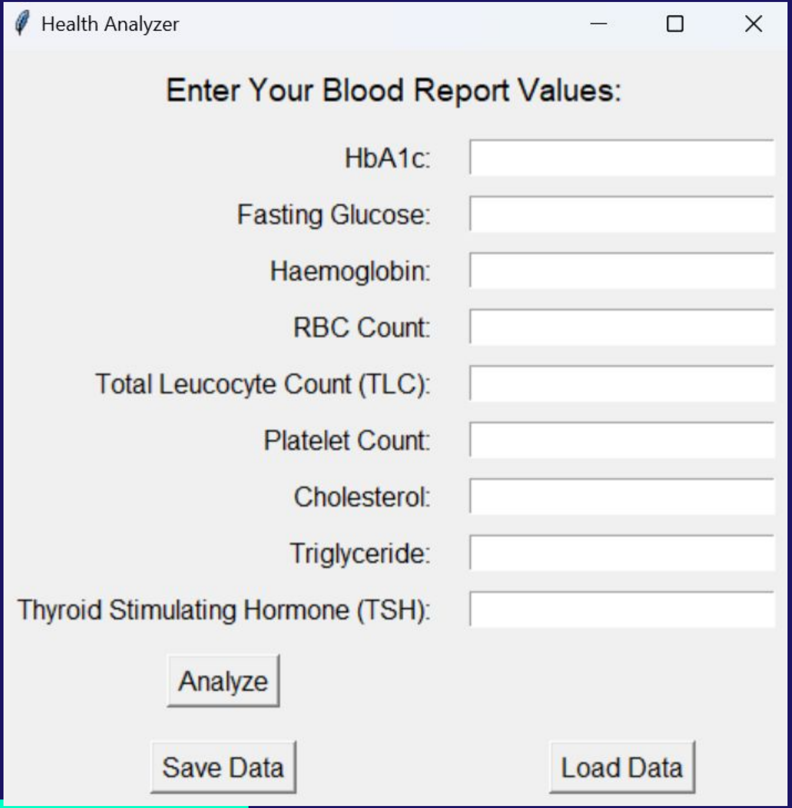
Error Handling

Robust error handling for invalid inputs.

Improved GUI

Interactive and user-friendly GUI.

DEMO



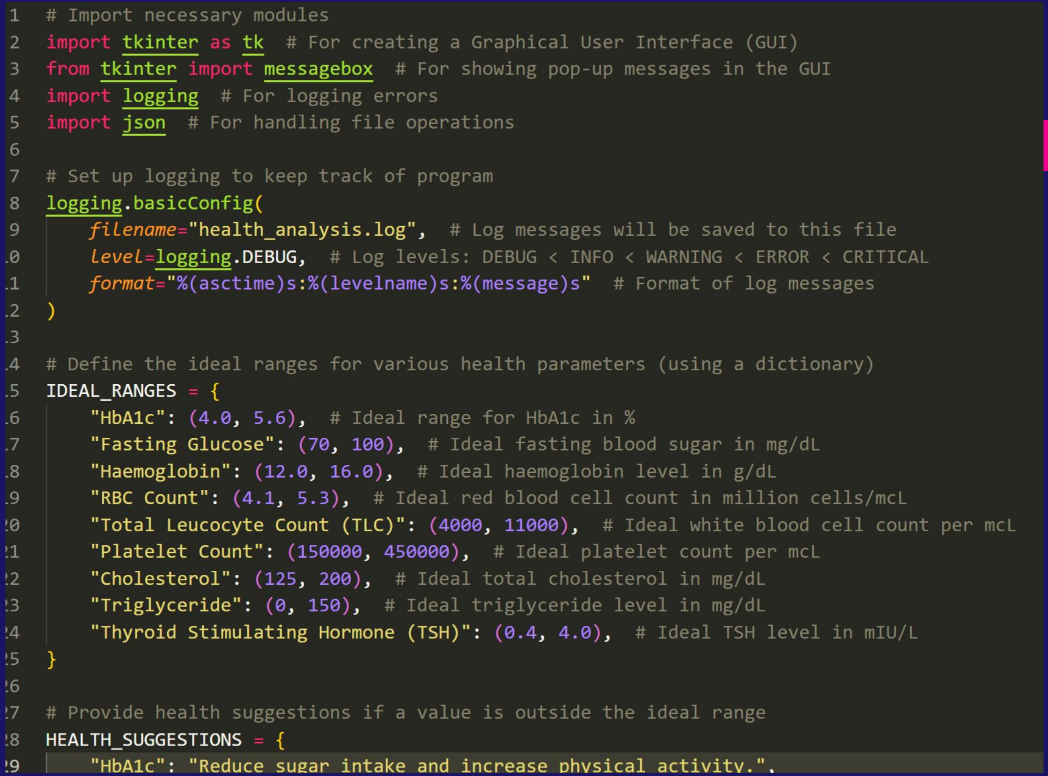
Health Analyzer

Enter Your Blood Report Values:

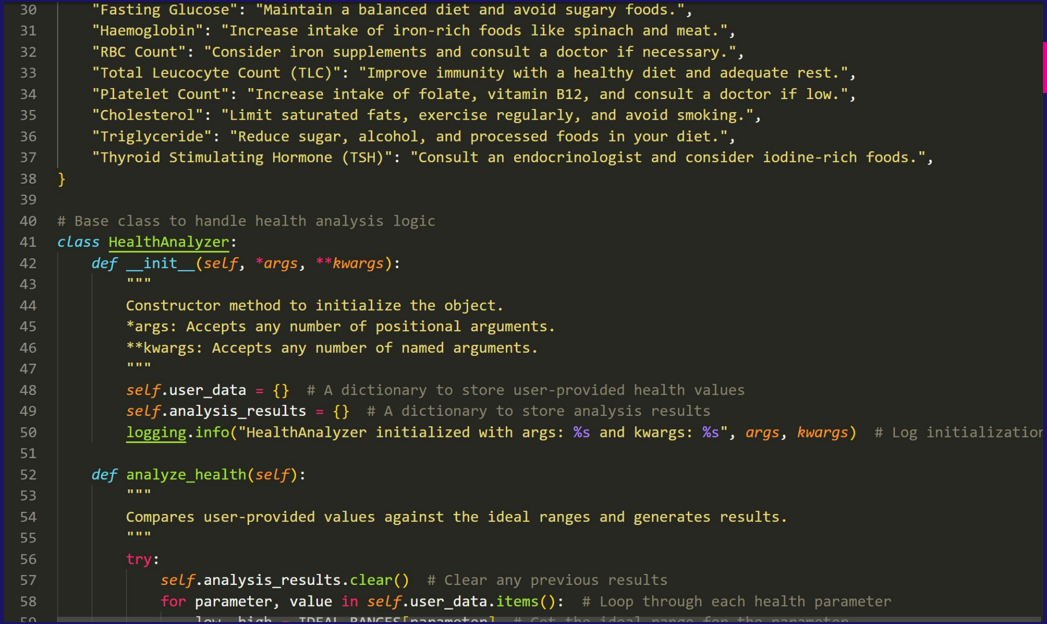
HbA1c:	
Fasting Glucose:	
Haemoglobin:	
RBC Count:	
Total Leucocyte Count (TLC):	
Platelet Count:	
Cholesterol:	
Triglyceride:	
Thyroid Stimulating Hormone (TSH):	

Here is the GUI made using tkinter. User enters their values for further analysis which can be then saved or loaded.

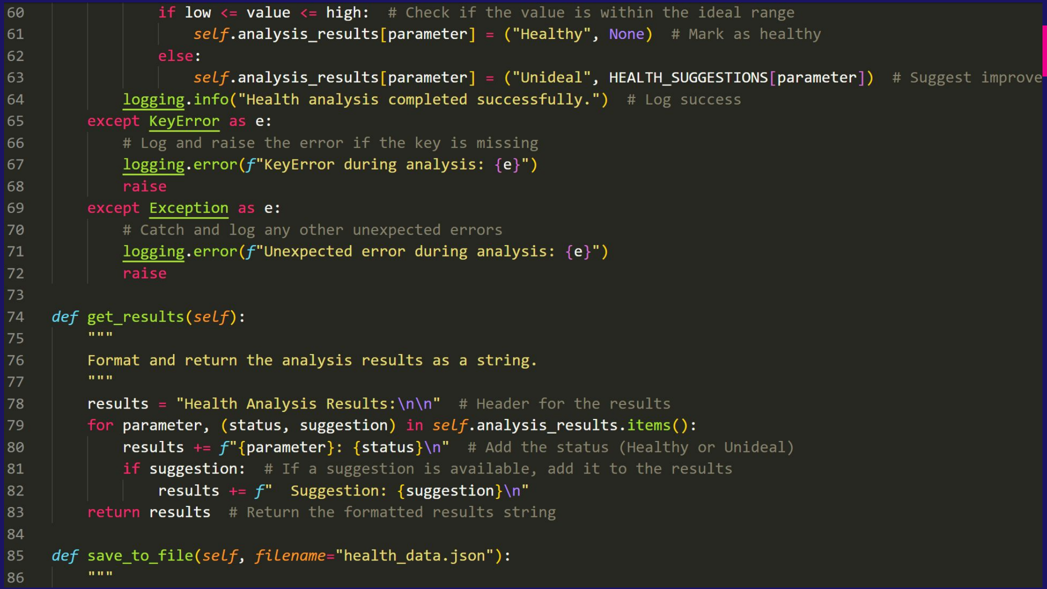
CODE



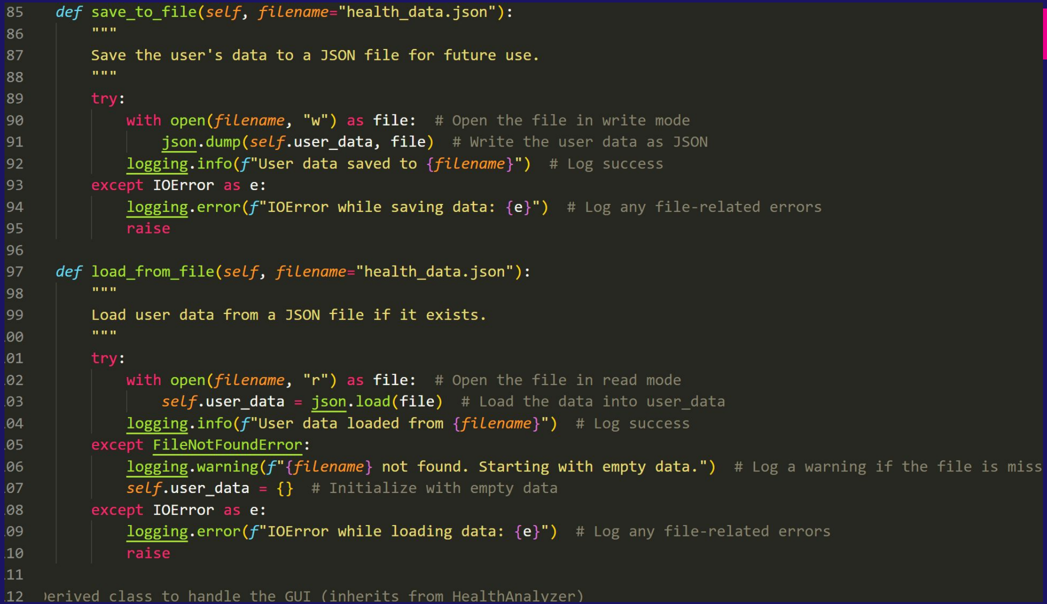
```python
# Import necessary modules
import tkinter as tk  # For creating a Graphical User Interface (GUI)
from tkinter import messagebox  # For showing pop-up messages in the GUI
import logging  # For logging errors
import json  # For handling file operations

# Set up logging to keep track of program
logging.basicConfig(
    filename="health_analysis.log",  # Log messages will be saved to this file
    level=logging.DEBUG,  # Log levels: DEBUG < INFO < WARNING < ERROR < CRITICAL
    format="%(asctime)s:%(levelname)s:%(message)s"  # Format of log messages
)

# Define the ideal ranges for various health parameters (using a dictionary)
IDEAL_RANGES = {
    "HbA1c": (4.0, 5.6),  # Ideal range for HbA1c in %
    "Fasting Glucose": (70, 100),  # Ideal fasting blood sugar in mg/dL
    "Haemoglobin": (12.0, 16.0),  # Ideal haemoglobin level in g/dL
    "RBC Count": (4.1, 5.3),  # Ideal red blood cell count in million cells/mcL
    "Total Leucocyte Count (TLC)": (4000, 11000),  # Ideal white blood cell count per mcL
    "Platelet Count": (150000, 450000),  # Ideal platelet count per mcL
    "Cholesterol": (125, 200),  # Ideal total cholesterol in mg/dL
    "Triglyceride": (0, 150),  # Ideal triglyceride level in mg/dL
    "Thyroid Stimulating Hormone (TSH)": (0.4, 4.0),  # Ideal TSH level in mIU/L
}

# Provide health suggestions if a value is outside the ideal range
HEALTH_SUGGESTIONS = {
    "HbA1c": "Reduce sugar intake and increase physical activity.",
```



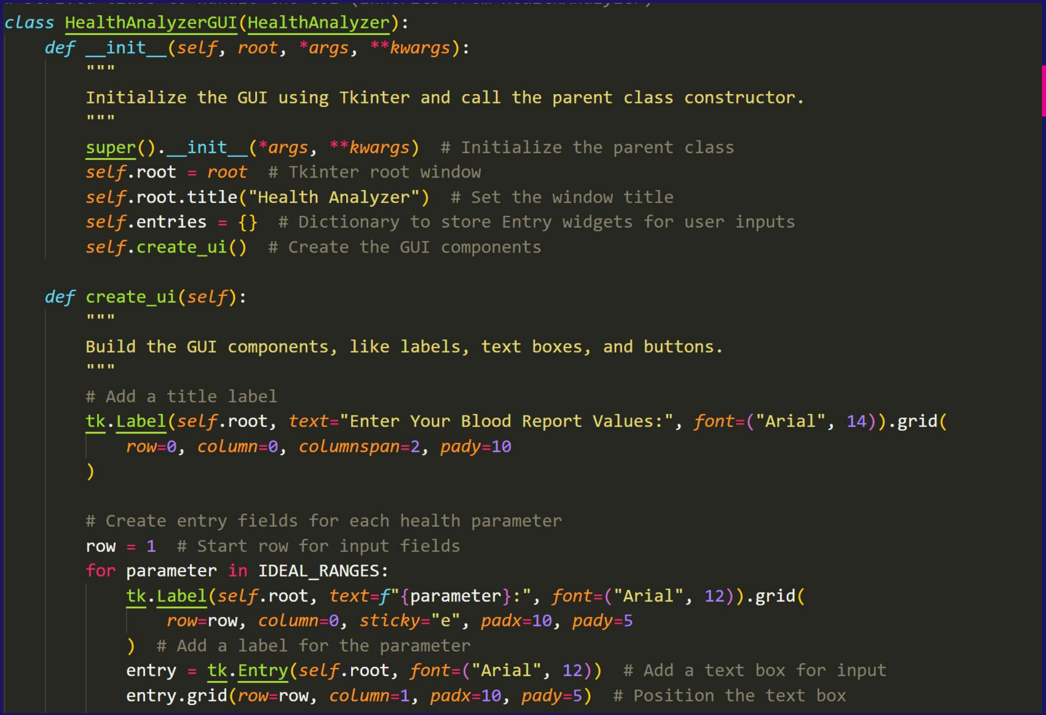
```python
    "Fasting Glucose": "Maintain a balanced diet and avoid sugary foods.",
    "Haemoglobin": "Increase intake of iron-rich foods like spinach and meat.",
    "RBC Count": "Consider iron supplements and consult a doctor if necessary.",
    "Total Leucocyte Count (TLC)": "Improve immunity with a healthy diet and adequate rest.",
    "Platelet Count": "Increase intake of folate, vitamin B12, and consult a doctor if low.",
    "Cholesterol": "Limit saturated fats, exercise regularly, and avoid smoking.",
    "Triglyceride": "Reduce sugar, alcohol, and processed foods in your diet.",
    "Thyroid Stimulating Hormone (TSH)": "Consult an endocrinologist and consider iodine-rich foods.",
}

# Base class to handle health analysis logic
class HealthAnalyzer:
    def __init__(self, *args, **kwargs):
        """
        Constructor method to initialize the object.
        *args: Accepts any number of positional arguments.
        **kwargs: Accepts any number of named arguments.
        """
        self.user_data = {}  # A dictionary to store user-provided health values
        self.analysis_results = {}  # A dictionary to store analysis results
        logging.info("HealthAnalyzer initialized with args: %s and kwargs: %s", args, kwargs)  # Log initializatio

    def analyze_health(self):
        """
        Compares user-provided values against the ideal ranges and generates results.
        """
        try:
            self.analysis_results.clear()  # Clear any previous results
            for parameter, value in self.user_data.items():  # Loop through each health parameter
                low, high = IDEAL_RANGES[parameter]  # Get the ideal range for the parameter
```



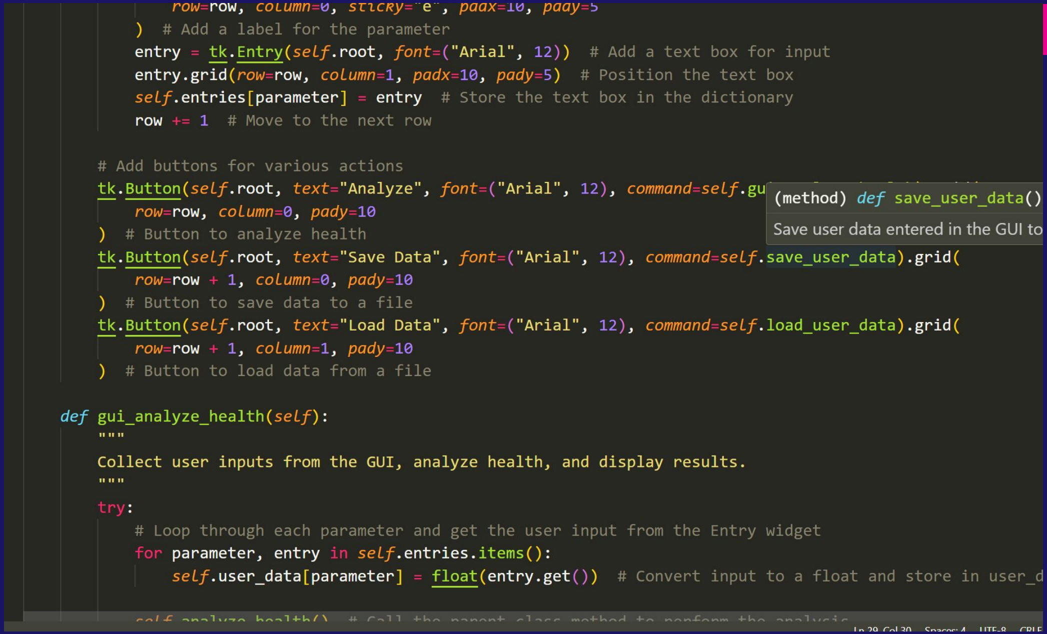
```python
                if low <= value <= high:  # Check if the value is within the ideal range
                    self.analysis_results[parameter] = ("Healthy", None)  # Mark as healthy
                else:
                    self.analysis_results[parameter] = ("Unideal", HEALTH_SUGGESTIONS[parameter])  # Suggest improve
            logging.info("Health analysis completed successfully.")  # Log success
        except KeyError as e:
            # Log and raise the error if the key is missing
            logging.error(f"KeyError during analysis: {e}")
            raise
        except Exception as e:
            # Catch and log any other unexpected errors
            logging.error(f"Unexpected error during analysis: {e}")
            raise

    def get_results(self):
        """
        Format and return the analysis results as a string.
        """
        results = "Health Analysis Results:\n\n"  # Header for the results
        for parameter, (status, suggestion) in self.analysis_results.items():
            results += f"{parameter}: {status}\n"  # Add the status (Healthy or Unideal)
            if suggestion:  # If a suggestion is available, add it to the results
                results += f"  Suggestion: {suggestion}\n"
        return results  # Return the formatted results string

    def save_to_file(self, filename="health_data.json"):
        """
```



```
    def save_to_file(self, filename="health_data.json"):
        """
        Save the user's data to a JSON file for future use.
        """
        try:
            with open(filename, "w") as file:  # Open the file in write mode
                json.dump(self.user_data, file)  # Write the user data as JSON
            logging.info(f"User data saved to {filename}")  # Log success
        except IOError as e:
            logging.error(f"IOError while saving data: {e}")  # Log any file-related errors
            raise

    def load_from_file(self, filename="health_data.json"):
        """
        Load user data from a JSON file if it exists.
        """
        try:
            with open(filename, "r") as file:  # Open the file in read mode
                self.user_data = json.load(file)  # Load the data into user_data
            logging.info(f"User data loaded from {filename}")  # Log success
        except FileNotFoundError:
            logging.warning(f"{filename} not found. Starting with empty data.")  # Log a warning if the file is miss
            self.user_data = {}  # Initialize with empty data
        except IOError as e:
            logging.error(f"IOError while loading data: {e}")  # Log any file-related errors
            raise

erived class to handle the GUI (inherits from HealthAnalyzer)
```



```python
class HealthAnalyzerGUI(HealthAnalyzer):
    def __init__(self, root, *args, **kwargs):
        """
        Initialize the GUI using Tkinter and call the parent class constructor.
        """
        super().__init__(*args, **kwargs)  # Initialize the parent class
        self.root = root  # Tkinter root window
        self.root.title("Health Analyzer")  # Set the window title
        self.entries = {}  # Dictionary to store Entry widgets for user inputs
        self.create_ui()  # Create the GUI components

    def create_ui(self):
        """
        Build the GUI components, like labels, text boxes, and buttons.
        """
        # Add a title label
        tk.Label(self.root, text="Enter Your Blood Report Values:", font=("Arial", 14)).grid(
            row=0, column=0, columnspan=2, pady=10
        )

        # Create entry fields for each health parameter
        row = 1  # Start row for input fields
        for parameter in IDEAL_RANGES:
            tk.Label(self.root, text=f"{parameter}:", font=("Arial", 12)).grid(
                row=row, column=0, sticky="e", padx=10, pady=5
            )  # Add a label for the parameter
            entry = tk.Entry(self.root, font=("Arial", 12))  # Add a text box for input
            entry.grid(row=row, column=1, padx=10, pady=5)  # Position the text box
```

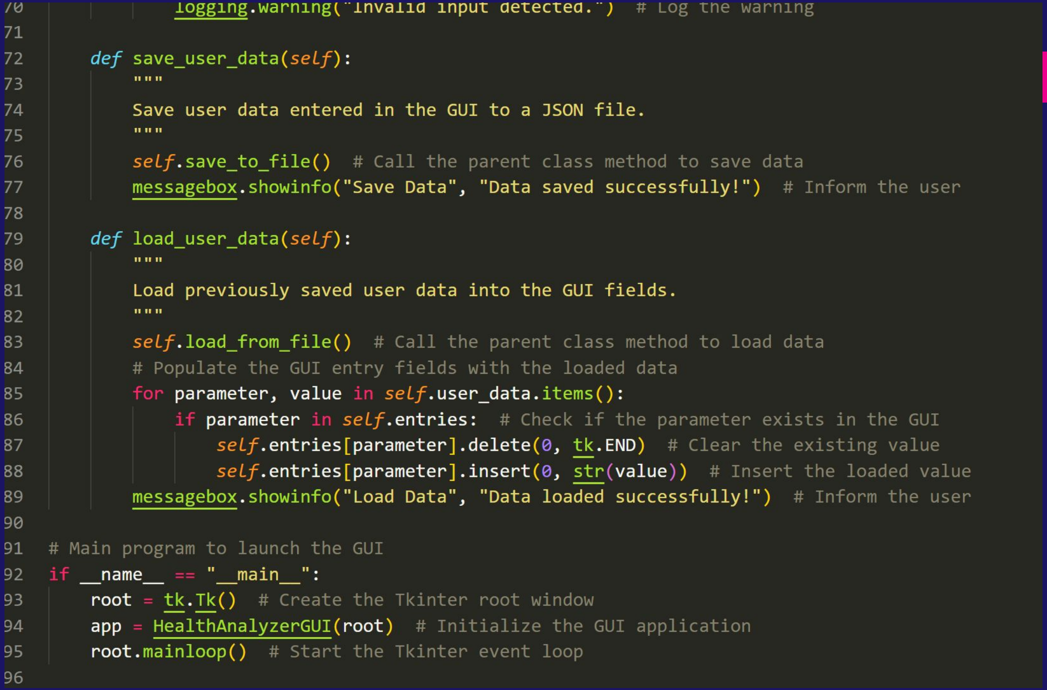


```
                row=row, column=0, sticky="e", padx=10, pady=5
            )  # Add a label for the parameter
            entry = tk.Entry(self.root, font=("Arial", 12))  # Add a text box for input
            entry.grid(row=row, column=1, padx=10, pady=5)  # Position the text box
            self.entries[parameter] = entry  # Store the text box in the dictionary
            row += 1  # Move to the next row

        # Add buttons for various actions
        tk.Button(self.root, text="Analyze", font=("Arial", 12), command=self.gu
            row=row, column=0, pady=10
        )  # Button to analyze health
        tk.Button(self.root, text="Save Data", font=("Arial", 12), command=self.save_user_data).grid(
            row=row + 1, column=0, pady=10
        )  # Button to save data to a file
        tk.Button(self.root, text="Load Data", font=("Arial", 12), command=self.load_user_data).grid(
            row=row + 1, column=1, pady=10
        )  # Button to load data from a file

    def gui_analyze_health(self):
        """
        Collect user inputs from the GUI, analyze health, and display results.
        """
        try:
            # Loop through each parameter and get the user input from the Entry widget
            for parameter, entry in self.entries.items():
                self.user_data[parameter] = float(entry.get())  # Convert input to a float and store in user_d

            self.analyze_health()  # Call the parent class method to perform the analysis
```

```
(method) def save_user_data()
Save user data entered in the GUI to
```

```
Ln 29, Col 30   Spaces: 4   UTF-8   CRLF
```

CODE



```python
            logging.warning("Invalid input detected.")  # Log the warning

    def save_user_data(self):
        """
        Save user data entered in the GUI to a JSON file.
        """
        self.save_to_file()  # Call the parent class method to save data
        messagebox.showinfo("Save Data", "Data saved successfully!")  # Inform the user

    def load_user_data(self):
        """
        Load previously saved user data into the GUI fields.
        """
        self.load_from_file()  # Call the parent class method to load data
        # Populate the GUI entry fields with the loaded data
        for parameter, value in self.user_data.items():
            if parameter in self.entries:  # Check if the parameter exists in the GUI
                self.entries[parameter].delete(0, tk.END)  # Clear the existing value
                self.entries[parameter].insert(0, str(value))  # Insert the loaded value
        messagebox.showinfo("Load Data", "Data loaded successfully!")  # Inform the user

# Main program to launch the GUI
if __name__ == "__main__":
    root = tk.Tk()  # Create the Tkinter root window
    app = HealthAnalyzerGUI(root)  # Initialize the GUI application
    root.mainloop()  # Start the Tkinter event loop

```

THANKS!