Software Design Specification Document

AI Calorie Predictor

Revision 1.0

Chaz Cornejo, Justin Liang, Samantha Erickson, Michael Pastora

https://github.com/JXAsain/Calories-Burnt-Prediction

Table of Contents

1. Introduction

1.1 Purpose

This program implements and capitalizes on machine learning to estimate the number of calories a user burns during physical activity, which is calculated from their basic information. The system features a PyQt6 interface. It supports visual comparisons using custom CSV files. This makes the application easily accessible to all users, providing a smooth experience that provides insight into one's health.

1.2 System Overview

The program combines a machine learning model with a user-friendly front-end to provide fast and accurate calorie burn predictions. It includes interactive graphs and file upload capabilities.

1.3 Design Map

The project integrates:

- XGBoost based prediction model
    - XGBoost was chosen as our machine learning algorithm because it was the best fit for our application. We tested several models: Linear Ridge Regression tended to underfit data while Random Forest tended to overfit. XGBoost had the best balance of both accuracy and generalization. What this machine learning algorithm does is combine a bunch of decision trees (smaller choices/mistakes) into building the most ideal path. This made it great for the variability in physical and nutritional data, ultimately providing us with the best outputs.
- A PyQt GUI for desktop apps.
- Matplotlib-based plots for personal and uploaded data.

1.4 Definitions and Acronyms

- **kcal**: Kilocalories
    - A unit of energy that measures the energy burned during exercise.

- **GUI**: Graphical User Interface
    - The parts of the application the user interacts with, this being the interface, buttons, text, etc.
- **CSV**: Comma Separated Values
    - A basic file format used for storing data tables, it separates values with a comma.
- **PyQt**: Python Qt Binding for GUI development
    - A Python library that provides tools to create desktop applications, including buttons, windows, and other aspects of a strong user interface.
- **ML**: Machine Learning
    - A form of artificial intelligence that provides computers with the ability to learn from data and make predictions. In this app, it's used to predict the calories burnt.

2. Design Considerations

2.1 Assumptions

- Users will run the application in a desktop environment.
- CSV files align with the required parameters for calorie data.
- Users can enter basic health metrics. (age, height, heart rate, etc.)

2.2 Constraints

- Platform is limited to a local system's power, rather than being cloud based.
- CSV formatting must align with backend expectations, it's not flexible to an improper format. (Format = User_ID, Gender, Age, Height, Weight, Duration, Heart_Rate, Body_Temp)

2.3 System Environment

- Python 3.10+

- Required libraries: PyQt6, matplotlib, pandas, joblib, scikit-learn, xgboost
- SQLite for user management

2.4 Risks and Volatile Areas

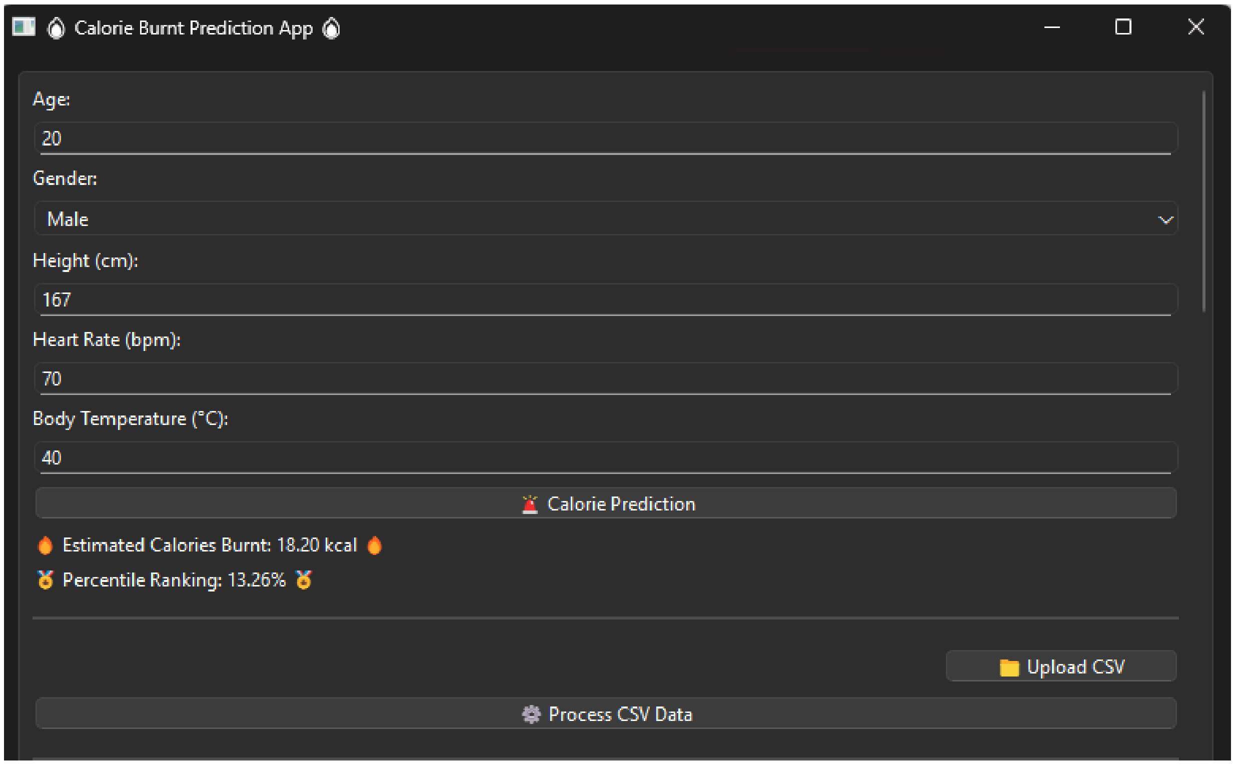
- Input validation and exception handling must be thorough.
- File upload errors or improper CSVs may break predictions.
- Backend model changes require updates to both GUIs.

3. High Level Design

3.1 Calorie Prediction Interface

Users input data (age, gender, height, heart rate, body temperature). After validation, the data is passed to the model, which returns the predicted number of calories burned, along with a percentile ranking pulled from a larger dataset. This ranking is further expanded through the creation of graphs for each category. Users can choose between visualizing their comparison with either a histogram or scatterplot.

3.1 Ex:



Calorie Burnt Prediction App

Age:

20

Gender:

Male

Height (cm):

167

Heart Rate (bpm):

70

Body Temperature (°C):

40

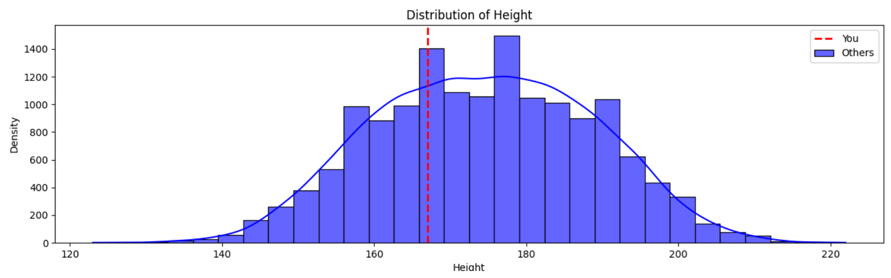
Calorie Prediction

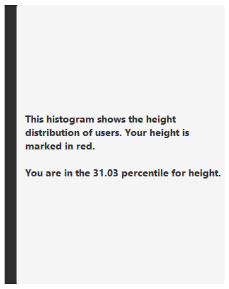
Estimated Calories Burnt: 18.20 kcal

Percentile Ranking: 13.26%

Upload CSV

Process CSV Data





This histogram shows the height distribution of users. Your height is marked in red.

You are in the 31.03 percentile for height.

Users are provided with explanations for each graph, providing more insight into their metrics. This ranges from simple affirmations to warnings depending on their data placement. The red line indicates where the user’s results stand in comparison to others pulled from a larger dataset.

4. Low Level Design

4.1 Backends

4.1.1 Model Prediction

- Function run(userData) returns predicted calorie burn. Backend model is loaded from calories_predictor.pkl. Input normalization is handled before prediction.

4.1.2 Plot Rendering

This application creates its graphs using a Python library called matplotlib. These graphs are useful as they expand the clarity of the information provided from each output.

- Userdata_compare_histogram(): This function is used to generate a **bar-style** graph, indicating where the user's data is placed among the larger dataset built into the app.
- Userdata_compare_statter(): This function is used to generate a **scatter** graph, where each dot is a person. The user's dot is highlighted to make them easily distinguished from the rest of the data.
- Received_csv_data_histogram(): This function is used to create **bar** charts from an uploaded CSV file. This means that bar graphs can be created from a dataset comprising a large group of people.
- Received_csv_data_scatter(): This function is used to create **scatter** charts from an uploaded CSV file. This means that scatter plots can be created from a dataset comprising a large group of people.

4.2 GUI

4.2.1 PyQt GUI

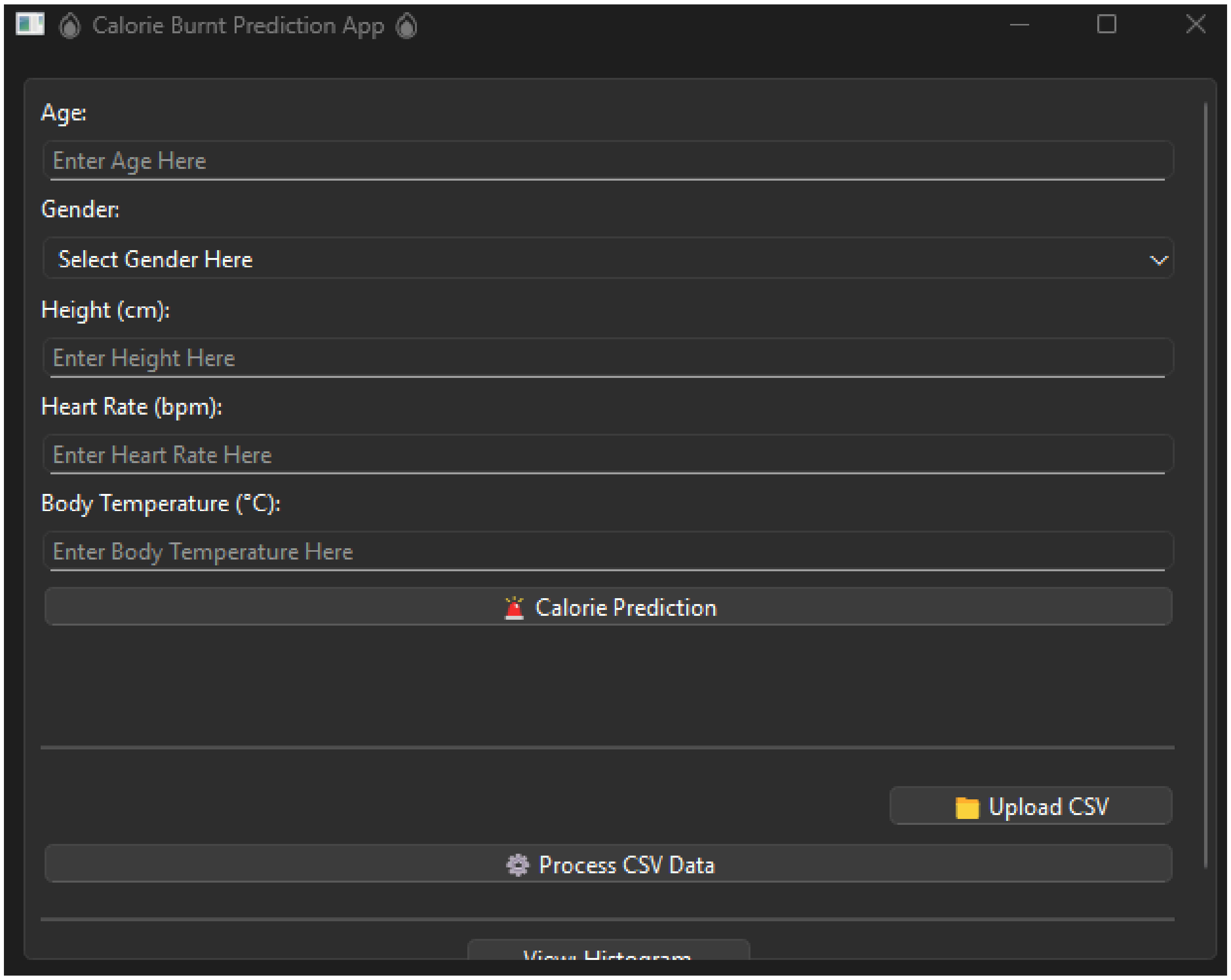
- Multi-screen window using PyQt6.
- Input validation ensures clean, expected data formats.
- Uses a modular design:
    - elementsUI.py builds all UI widgets and layout.

    - eventHandlers.py manages logic for prediction, plotting, CSV loading, and user feedback.
    - main.py dynamically connects UI and logic via extend_caloriesPredictor().
- Matplotlib plots are embedded with interactive visual feedback.
- Descriptive explanations accompany user-generated plots for added clarity.
- Scrollable interface with expandable layouts for enhanced usability.

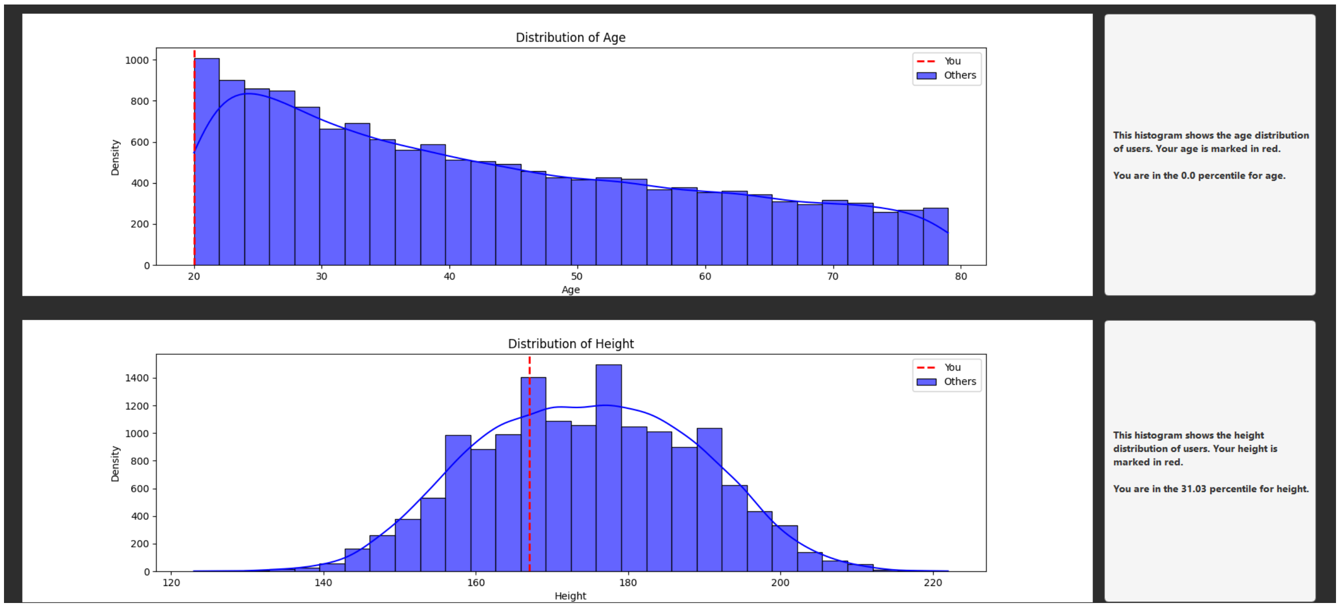
5. User Interface Design

5.1 Screen 1: Calorie Prediction

- Inputs for:
    - Age
    - Gender
    - Height (cm)
    - Heart Rate (bpm)
    - Body Temperature (°C)
- "Calorie Prediction" button runs program.
- Prediction output shown with percentile ranking.



- Toggle button allows switching between histogram and scatter plot.
- Descriptive explanations adapt based on user data:



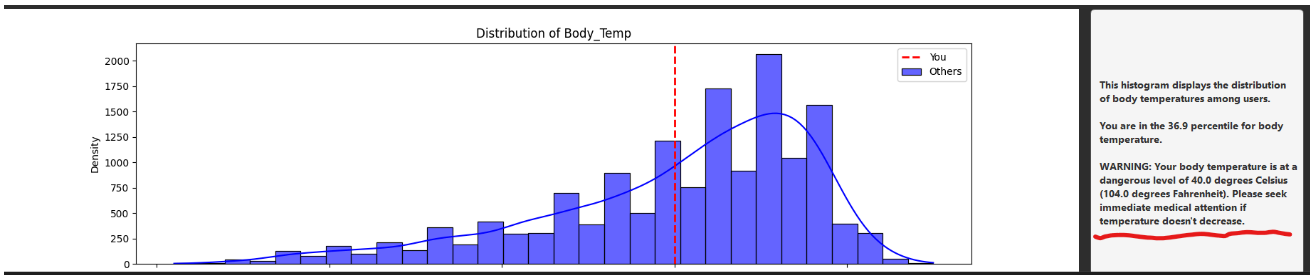
This histogram shows the age distribution of users. Your age is marked in red.

You are in the 0.0 percentile for age.

This histogram shows the height distribution of users. Your height is marked in red.

You are in the 31.03 percentile for height.

- Health warnings which depend on the user's heart rate/temperature.



This histogram displays the distribution of body temperatures among users.

You are in the 36.9 percentile for body temperature.

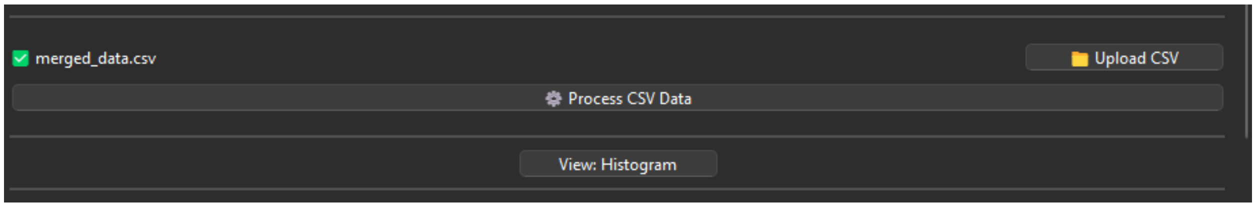
WARNING: Your body temperature is at a dangerous level of 40.0 degrees Celsius (104.0 degrees Fahrenheit). Please seek immediate medical attention if temperature doesn't decrease.

- Feedback comparing user stats to global dataset

5.2 Screen 3: CSV Upload and Visualization

To expand the application's use cases, it also features CSV uploading capabilities. This has multiple uses, as it can be used to view trends in large datasets. An example of this being used in real world applications would be a hospital using it to visualize information on their patients.

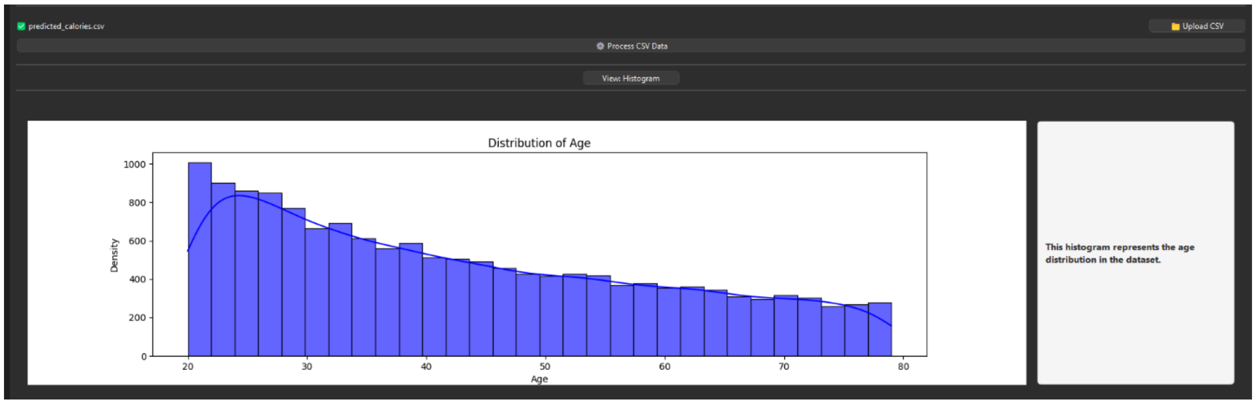
- File upload for custom .csv datasets using QFileDialog.
- The "Process CSV Data" button triggers plot generation from the uploaded file.



^ Example of CSV loaded into application.

- Displays age, height, heart rate, and temperature distributions or comparisons.

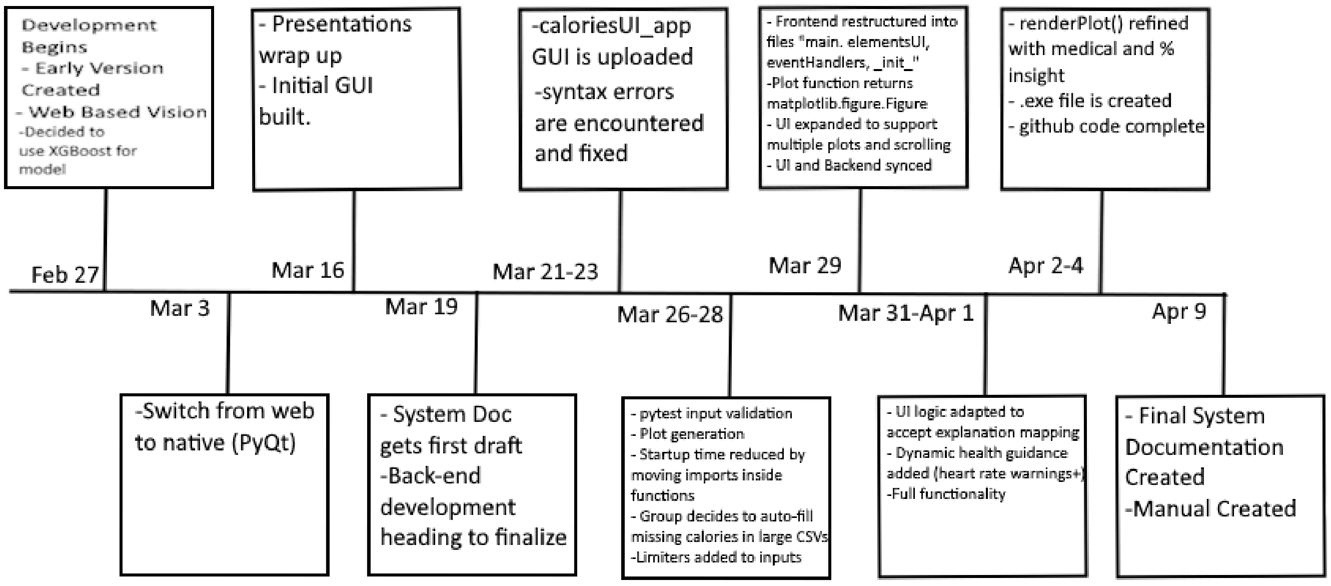
  Ex: Age output from CSV -



This histogram represents the age distribution in the dataset.

- Visuals adapt to histogram or scatterplot toggle.
- Each graph is accompanied by an explanation, giving users an explanation of the data.
- Layout uses scrollable vertical alignment with full-width support for large visual content.

Appendix A: Project Timeline


Feb 27
Development Begins
- Early Version Created
- Web Based Vision
-Decided to use XGBoost for model
Mar 3
-Switch from web to native (PyQt)
Mar 16
- Presentations wrap up
- Initial GUI built.
Mar 19
- System Doc gets first draft
-Back-end development heading to finalize
Mar 21-23
-caloriesUI_app GUI is uploaded
-syntax errors are encountered and fixed
Mar 26-28
- pytest input validation
- Plot generation
- Startup time reduced by moving imports inside functions
- Group decides to auto-fill missing calories in large CSVs
-Limiters added to inputs
Mar 29
- Frontend restructured into files "main. elementsUI, eventHandlers, _init_"
-Plot function returns matplotlib.figure.Figure
- UI expanded to support multiple plots and scrolling
- UI and Backend synced
Mar 31-Apr 1
- UI logic adapted to accept explanation mapping
- Dynamic health guidance added (heart rate warnings+)
-Full functionality
Apr 2-4
- renderPlot() refined with medical and % insight
- .exe file is created
- github code complete
Apr 9
- Final System Documentation Created
-Manual Created

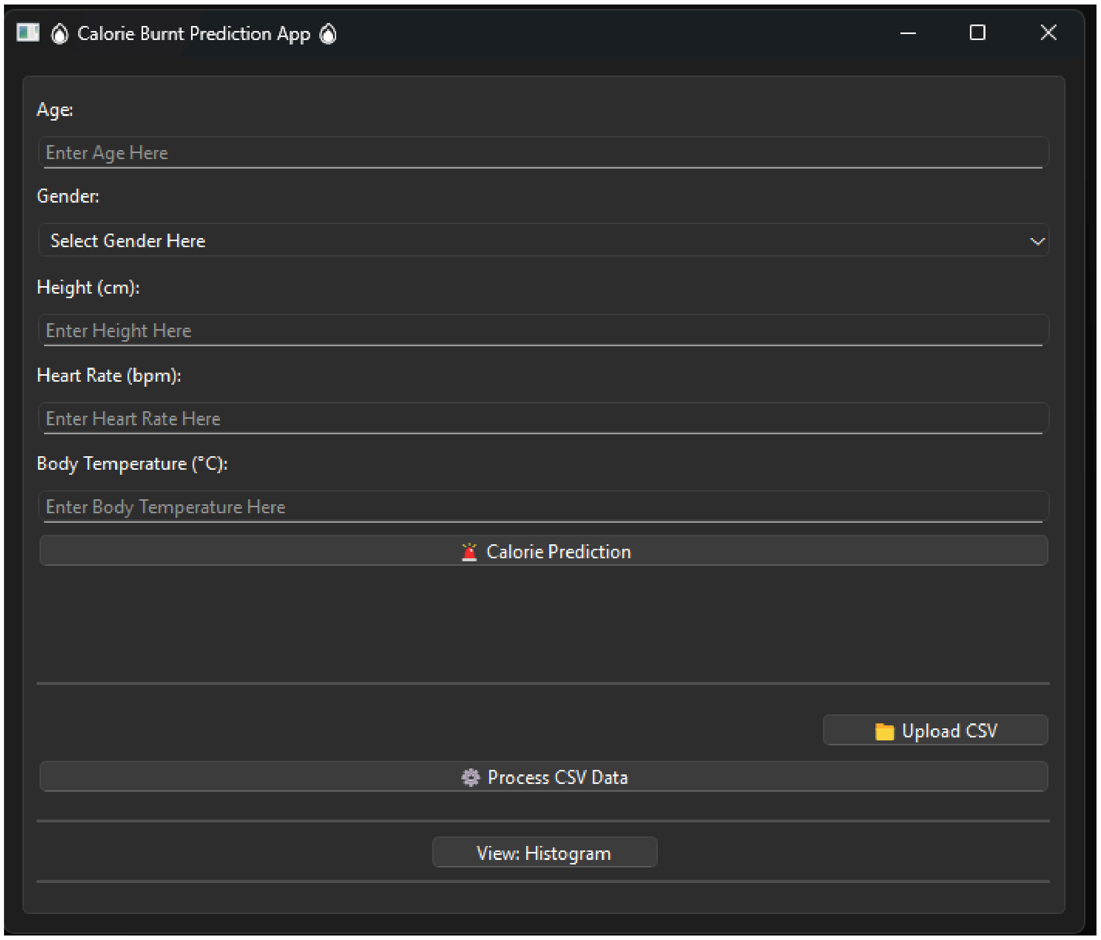
Appendix B: Manual

How to Use the Application:

Step 1: Enter Your Information

Fill in the following fields at the top of the app window:

- **Age:** Your age in years
- **Gender:** Select Male or Female from the dropdown
- **Height (cm):** Your height in centimeters
- **Heart Rate (bpm):** Your current heart rate in beats per minute
- **Body Temperature (°C):** Your body temperature in Celsius



Make sure all fields are filled in accurately.

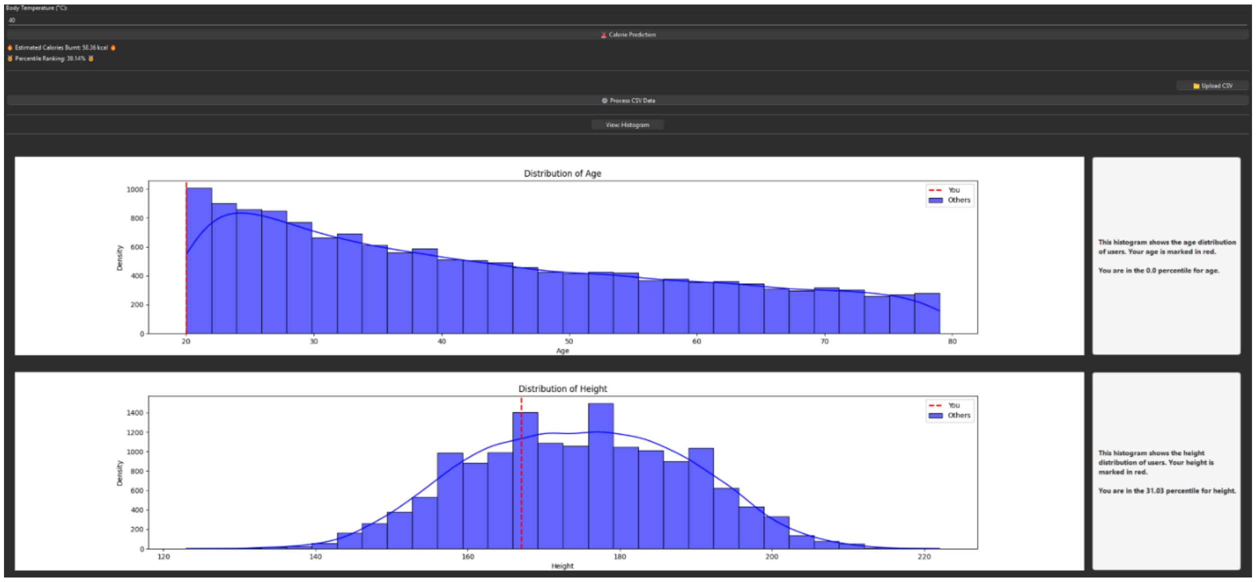
Step 2: Predict Your Calories

Click the **Calorie Prediction** button. The app will:

- Run your data through a machine learning model
- Predict the number of calories you likely burned
- Show your **Estimated Calories Burnt**
- Show your **Percentile Ranking** compared to others

Percentile Ranking helps you understand how your activity compares to others. A higher percentile means you're burning more calories than most users.

Example:



🔥 Estimated Calories Burnt: 58.36 kcal 🔥

🏅 Percentile Ranking: 39.14% 🏅

This histogram shows the age distribution of users. Your age is marked in red.

You are in the 0.0 percentile for age.

This histogram shows the height distribution of users. Your height is marked in red.

You are in the 31.03 percentile for height.

Step 3: Upload a CSV (Optional)

To compare group results or test data from external sources:

1. Click the **Upload CSV** button
2. Select a .csv file formatted with the correct headers (Age, Height, Heart_Rate, Body_Temp, etc.)
3. The filename will appear with a **checkmark** if successfully loaded

Step 4: Process CSV and View Graphs

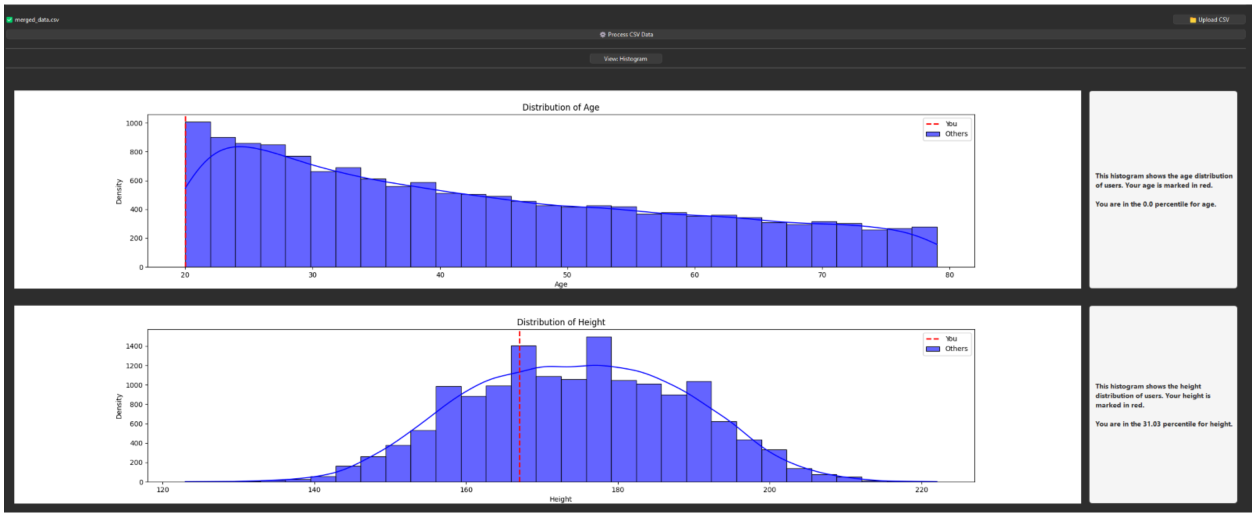
Once your CSV is uploaded:

- Click **Process CSV Data**
- Click the **View: Histogram** toggle to switch between **Histogram** and **Scatter Plot** diagrams
- The app will display graphs showing distributions and comparisons based on:
  - Age
  - Height
  - Heart Rate
  - Body Temperature

Your own data will be marked clearly in red (dotted line or red marker) in each graph.

Potential Problems:

- **Error loading CSV**: Check that the file is in .csv format and follows the specified format.
- **Invalid Input**: Make sure all fields are filled in with numbers and within the allowed range.
- **Graphs not showing**: Make sure the prediction step is complete, and your input includes valid calorie data.



This histogram shows the age distribution of users. Your age is marked in red.

You are in the 0.0 percentile for age.

This histogram shows the height distribution of users. Your height is marked in red.

You are in the 31.03 percentile for height.

Appendix C: Challenges Faced with Graph Display Integration

Challenge: Displaying Matplotlib Figures Within a Scrollable PyQt6 Interface

One of our more technical challenges was ensuring Matplotlib-generated figures were rendered correctly and consistently within a PyQt6 GUI, particularly when integrating a scrollable interface and accommodating dynamic user inputs.

The Problems

1. **Figures Were Cut Off or Cropped**

When embedding FigureCanvas widgets inside the layout, the plots frequently appeared truncated, with only partial content visible. Despite using `setMinimumHeight()` or `setFixedSize(),` the scroll area often restricted the complete rendering of the figure.

2. **Layout Constraints in Scroll Areas**

The scrollable region (QScrollArea) often conflicted with size policies. Our initial approach of using `setMinimumHeight()` on content widgets caused the layout to behave rigidly, preventing figures from resizing or scrolling properly when overflow occurred.

3. **Overlapping and Persistent Canvases**

While implementing multiple plot renderings, which included CSV data and individual user predictions, we faced challenges with new canvases overlapping previous ones. This situation required us to carefully track and clean up old widgets before rendering new ones.

4. **Canvas Not Resizing With the Window**

At one point, switching to `showMaximized()` caused the application window to appear full screen. However, the canvas height did not adjust accordingly, leading to a mismatch between the window size and the figure display. We resolved this by using `setSizePolicy(Expanding, Expanding)` and removing hard-coded height constraints.

5. **Packaging Modules and Files During .exe Creation**

When making an .exe file, all used packages and files must be included for it to run as a standalone application. Packaging PyQt6 had issues because of conflicts with PyQt5 which was in the environment as well. XGBoost also does not package itself directly, so there would be issues with the file running properly due to this module not being handled correctly. Lasty, all the files for data that are used within the application are manually specified for being included within the file because situations where they were attempted to be referenced to externally lead to crashes.

Lessons Learned

- PyQt's layout system is sensitive to widget sizing, especially within scrollable areas.
- Hard-coding size constraints, such as `setMinimumHeight(),` can result in layout rigidity and visual issues.
- Always remove old canvas widgets before adding new ones to prevent overlap or memory leaks.
    - Utilize size policies rather than fixed sizes for more responsive behavior, allowing PyQt to manage the layout naturally.
- .spec files are what build executable files; these files can be edited directly and ran for flexible creation.