

Title: Heart Attack Risk Assessment System
Technology: Python (Console Application, CSV Storage)
Name: Aman Nawaz
Date: 24-11-2025

Introduction

This application helps users quickly estimate their heart attack risk using common clinical and lifestyle factors such as age, blood pressure, cholesterol, diabetes, smoking, and physical activity. It aims to raise awareness and encourage medical consultation, not to replace professional diagnosis. [cdr.lib.unc+1](#)

Problem statement

Heart disease is a leading cause of mortality, and many people are unaware of their own risk level until severe symptoms occur. There is a need for a simple, easily accessible tool that can give a rough indication of risk based on self-reported information. [github+1](#)

Functional requirements

- Collect user inputs: age, chest pain, systolic and diastolic blood pressure, cholesterol, diabetes, family history, physical activity, obesity, and smoking status. [pythonhosted](#)
 - Validate basic numeric inputs for age and blood pressure; handle invalid input with an error message and exit safely. [py-template.readthedocs](#)
 - Compute a risk score using fixed rules and classify users as Low, Moderate, or High risk. [pythonhosted](#)
 - Display an appropriate text message according to the risk level. [pythonhosted](#)
 - Store each assessment as a row in a CSV file (`heart_risk_data.csv`) with key fields and risk level. [py-template.readthedocs](#)
-

Non-functional requirements

- Usability: Questions must be clear and answerable with simple yes/no or numeric values via the console. [pythonhosted](#)
- Reliability: The program should handle basic input errors without crashing and consistently write valid rows to the CSV file. [py-template.readthedocs](#)

- Portability: The script should run on any system with Python installed and access to the filesystem.[py-template.readthedocs](#)
 - Maintainability: The logic is organized into a single function (`heart_attack_check`) to simplify understanding and updates.[py-template.readthedocs](#)
-

System architecture

The architecture is a simple single-tier console application. There are three logical layers inside the script:[py-template.readthedocs](#)

- Presentation: console input and output (`input`, `print`).
 - Business logic: risk score calculation and risk-level classification.
 - Data layer: CSV file operations using the `csv` and `os` modules.[py-template.readthedocs](#)
-

Design diagrams

Below are textual descriptions you can later convert into proper diagrams in a modeling tool.

Use case diagram

Actors:

- User (patient or general user)

Main use cases:

- Enter health details.
- View heart attack risk level.
- Persist assessment to CSV storage.

Relationships:

- The User initiates “Enter health details” which includes providing all answers.[pythonhosted](#)
- After data entry, the system performs “Calculate risk” and then “Display result” and “Save assessment”.[pythonhosted](#)

Workflow diagram

1. Start program (`__main__` calls `heart_attack_check`).
2. Display welcome and instructions.[pythonhosted](#)
3. Prompt sequentially for age, chest pain, blood pressure, cholesterol, diabetes, family history, physical activity, obesity, and smoking.[pythonhosted](#)

4. If a numeric conversion fails, show “Invalid input” and terminate.[py-template.readthedocs](#)
5. Initialize `risk_score = 0` and apply each rule to accumulate the score.[pythonhosted](#)
6. Map score to risk level: Low / Moderate / High.[pythonhosted](#)
7. Print risk level and safety message.[pythonhosted](#)
8. Prepare row data, check if CSV exists, write header if not, then append the row.[py-template.readthedocs](#)
9. Display confirmation that data is saved; end program.[py-template.readthedocs](#)

Sequence diagram

Objects: User, heart_attack_check, Console I/O, File System.

- User → heart_attack_check: start.
- heart_attack_check → Console: display welcome and questions; Console → User: prompts; User → Console: answers.[pythonhosted](#)
- heart_attack_check: compute `risk_score` and risk level.[pythonhosted](#)
- heart_attack_check → Console: display result.[pythonhosted](#)
- heart_attack_check → File System: open CSV, write header (if needed), append row, close file.[py-template.readthedocs](#)
- heart_attack_check → User: message “data saved”, then terminate.[py-template.readthedocs](#)

Class / component diagram

Components:

- heart_attack_check function: encapsulates all logic.[pythonhosted](#)
- csv module: provides `DictWriter` for structured CSV writing.[py-template.readthedocs](#)
- os module: provides `path.exists` to check CSV existence.[py-template.readthedocs](#)

Potential future classes (not yet implemented):

- PatientInput to hold user data.
- RiskCalculator to compute risk score.
- CsvRepository to abstract storage.[github](#)

ER diagram (CSV storage)

Since data is stored in a single CSV, conceptually it maps to one entity:

Entity: Assessment

Attributes: Age, Chest Pain, Systolic BP, Diastolic BP, Cholesterol, Diabetes, Family History, Physical Activity, Obesity, Smoking, Risk Level.[py-template.readthedocs+1](#)

Primary key: none enforced in code; each row is an independent record.[py-template.readthedocs](#)

Design decisions & rationale

- Rule-based scoring instead of machine learning to keep logic transparent and easy to adjust.[github](#)
 - Threshold-based classification (e.g., ≥ 10 = High) for straightforward interpretation and tuning.[pythonhosted](#)
 - CSV storage was chosen to avoid database setup and keep the project lightweight and portable.[py-template.readthedocs](#)
 - Single function structure simplifies reading for beginners, at the cost of limited modularity.[py-template.readthedocs](#)
-

Implementation details

Language and libraries: Python with `csv` and `os` from the standard library.[py-template.readthedocs](#)

Key implementation points:

- Input parsing uses `int()` for age and blood pressure; invalid inputs trigger a `ValueError` handled by a `try/except` block.[py-template.readthedocs](#)
- Risk scoring logic:
 - Age > 45: +2
 - Chest pain: +3
 - Systolic BP > 130 or Diastolic BP > 80: +2
 - Cholesterol > 200: +2
 - Diabetes: +2
 - Smoking: +2
 - Family history: +1
 - No physical activity: +1
 - Obesity: +2[pythonhosted](#)
- Risk classification:
 - `risk_score >= 10`: High Risk
 - `5 <= risk_score < 10`: Moderate Risk
 - `< 5`: Low Risk[pythonhosted](#)
- CSV writing uses `DictWriter` with `writeheader()` executed only when the file does not yet exist.[py-template.readthedocs](#)

Note: The `fieldnames` list and the row dictionary contain some naming mismatches (Cholestrol vs Cholesterol, Family Activity vs Family History), which should be corrected to ensure all columns are written correctly.[py-template.readthedocs](#)

Testing approach

Suggested tests:

- Valid-input tests: multiple combinations leading to Low, Moderate, and High risk to check score boundaries.[pythonhosted](#)
- Invalid-input tests: enter non-numeric values for age or BP to confirm that the error message appears and the program exits gracefully.[py-template.readthedocs](#)
- File tests:
 - First run: CSV is created and header is written.
 - Subsequent runs: header is not duplicated and rows are appended correctly.[py-template.readthedocs](#)

Testing is manual, but simple unit tests could be added around a refactored `calculate_risk_score()` function in the future.[github](#)

Challenges faced

Typical challenges for this type of project include:

- Designing a risk scoring scheme that is simple yet clinically sensible, understanding that it is only an approximation.[cdr.lib.unc](#)
 - Handling user input errors in a console environment while keeping the code readable.[py-template.readthedocs](#)
 - Ensuring data consistency between `fieldnames` and the dictionary keys when writing to CSV.[py-template.readthedocs](#)
-

Learnings & key takeaways

- Even a small Python script can implement a complete mini-system: input, processing, output, and persistent storage.[py-template.readthedocs](#)
 - Clear structuring of risk rules improves maintainability and helps users and reviewers understand how the score is computed.[pythonhosted](#)
 - Validating input and handling exceptions is essential for a reliable user-facing tool.[py-template.readthedocs](#)
-

Future enhancements

- Refactor into multiple functions or classes (e.g., separate modules for input, risk calculation, and storage).[github](#)

- Add a graphical interface or web front-end for better usability.[github](#)
 - Integrate more detailed medical guidelines or evidence-based risk models.[cdr.lib.unc](#)
 - Add authentication and a proper database to manage multiple users and longitudinal data.[github](#)
 - Implement unit tests and logging for better quality assurance.[github](#)
-

Reference

- General project structuring ideas and documentation patterns inspired by Python project templates and documentation guides.[py-template.readthedocs](#)
 - Concepts of risk assessment workflows and health risk scoring from healthcare risk assessment literature and tools.[cdr.lib.unc+1](#)
1. <https://pythonhosted.org/Euphorie/manuals/creation-guide.html>
 2. <https://www.scribd.com/document/728446850/Security-Risk-Assessment-Report-Template-2024>
 3. <https://github.com/yjhuang1119/Risk-assessment-model>
 4. <https://github.com/mmfarabi/Hawaii-Health-Risk-Assessment-System>
 5. <https://pyhealth.readthedocs.io/en/archived/>
 6. <https://pythonhosted.org/Euphorie/>
 7. <https://cdr.lib.unc.edu/downloads/hq37vz18s?locale=en>
 8. <https://pmc.ncbi.nlm.nih.gov/articles/PMC7169420/>
 9. https://py-template.readthedocs.io/_/downloads/en/latest/pdf/
 10. <https://github.com/SoumyoNathTripathy/RiskX>