Project Documentation

Project Title: Heart Disease Prediction System using Machine Learning

Team ID: NHA-011

Course / Track: AI & Data Science – Healthcare Project

Supervisor: Aya Abdullah

Date: 2025

1. Project Overview

Heart disease is one of the main causes of death worldwide. Early prediction of high-risk patients allows doctors to intervene earlier and improve the outcome. The goal of this project is to build a machine learning–based system that predicts whether a patient is likely to have heart disease based on several clinical features (age, blood pressure, cholesterol, chest pain type, etc.).

The system consists of a trained ML model, a backend API that exposes the model as a service, and a simple web interface where users can enter patient data and receive a prediction. The project follows the typical ML lifecycle: data understanding, preprocessing, model development, evaluation, and deployment as a small end-to-end application.

1.1 Objectives & Scope

- Build a binary classifier that predicts the presence or absence of heart disease.
- Analyse and preprocess a real heart disease dataset.
- Compare different machine learning algorithms and choose the best model.
- Expose the final model through a backend API for real-time predictions.
- Develop a simple, user-friendly interface for clinicians or users to interact with the model.
- Log predictions for basic monitoring and possible future improvements.

1.2 Stakeholders

- End users / clinicians: Use the system to quickly estimate heart disease risk for a patient.
- Project supervisors / instructors: Evaluate the technical quality and documentation of the project.
- Development team (NHA-011): Responsible for data analysis, model training, implementation and testing.

2. Literature Review & Background

Many research papers and Kaggle projects use classical machine learning algorithms such as Logistic Regression, Support Vector Machines, Decision Trees, Random Forests and Gradient Boosting for heart disease prediction. These models work well on structured tabular data because they can model non-linear relationships and interactions between features.

A common public dataset for this task is the UCI Heart Disease dataset, which contains demographic and clinical attributes. Previous work shows that with careful preprocessing and tuning, models can achieve good accuracy and recall, making them suitable as decision-support tools for doctors. Our project follows the same idea but focuses more on the full pipeline: from data and model to an actual working application with an API and UI.

3. Requirements Gathering & Analysis

3.1 Functional Requirements

- The system shall allow the user to input patient data (age, sex, chest pain type, blood pressure, cholesterol, etc.).
- The system shall send the input data to the backend API and receive a prediction.
- The system shall display the prediction result clearly (heart disease: yes / no).
- The backend shall load the trained ML model from a serialized file (.pkl).
- The backend shall validate the input data and handle missing or invalid values gracefully.
- The system shall log each prediction request and its result in a CSV file for analysis.

3.2 Non-Functional Requirements

- Usability: The web interface should be simple and clear for non-technical users.
- Performance: The system should return predictions within a few seconds.
- Reliability: The model file must be loaded correctly and reused without frequent failures.
- Maintainability: Code should be modular and reasonably documented to allow future changes.
- Portability: The system should run on a standard Python environment on a personal computer.

3.3 Dataset Requirements

We required a dataset that contained enough clinical attributes, a clear binary target label, and a reasonable number of records to train and evaluate machine learning models. The dataset had to be publicly available and allowed for educational use.

4. System Design

The system follows a simple client–server architecture. The UI acts as a client, sending patient data to the backend API. The API loads the ML model, preprocesses the data, and returns a prediction. A CSV file can be used as a lightweight log store for the predictions.

4.1 High-Level Architecture

- User Interface (ui.py): Collects input data from the user and displays the prediction.
- Backend API (app.py): Receives HTTP requests, validates and preprocesses data, loads the model and generates predictions.
- ML Model (best_model_optimized.pkl): Serialized model produced during the training phase.
- Training Notebook (final_depi_mil4.ipynb): Contains data loading, EDA, feature engineering and model training process.
- Logs (prediction_logs.csv): Stores input features and prediction outputs for monitoring.

4.2 Data Model & Storage

The main data sources in the system are the training dataset (e.g., heart_disease_dataset.csv) and the prediction logs (prediction_logs.csv). Each record corresponds to one patient with all the clinical attributes plus the target label during training. During inference, logs contain the input features and the predicted class.

4.3 Data Flow

1. The user opens the UI and fills in the patient information.
2. The UI sends the data as a request to the backend API.
3. The API preprocesses the data to match the format used during training.
4. The preprocessed data is passed to the loaded ML model.
5. The model predicts whether the patient is likely to have heart disease.
6. The API returns the prediction to the UI and optionally logs the request.
7. The UI displays the result to the user.

5. Implementation & Development

The project is implemented in Python. Development was done using Jupyter Notebook for experiments and Visual Studio Code for application code. Git and GitHub were used for version control and collaboration.

5.1 Tools & Technologies

- Programming Language: Python
- Libraries: numpy, pandas, scikit-learn, (xgboost if used), matplotlib, seaborn
- Backend: FastAPI or Flask style API implemented in app.py
- Frontend: Simple Python/Streamlit or custom UI implemented in ui.py
- Source Control: Git & GitHub

5.2 Code Structure

- app.py – Backend API for serving predictions.
- ui.py – User interface that sends patient data to the API.
- final_depi_mil4.ipynb – Notebook for data analysis and model training.
- best_model_optimized.pkl – Final trained model used in production.
- final_heart_model.pkl – Older/baseline model kept for reference.
- prediction_logs.csv – File for saving predictions during usage.
- README.md – Instructions for running the project and understanding the repository structure.

6. Testing & Evaluation

Testing was performed on two levels: model evaluation and basic functional testing of the application. For the model, we split the dataset into training and testing sets and evaluated

different algorithms using standard classification metrics. For the application, we manually tested the end-to-end flow from the UI to the API.

6.1 Model Evaluation

Several models were trained and compared (e.g., Logistic Regression, Random Forest, XGBoost). The final selected model was chosen based on its accuracy and especially its recall (sensitivity), because missing a real heart disease case is more critical than predicting some false positives.

Example metrics (to be updated with the actual numbers from the notebook):

- Accuracy: [XX.X]%
- Precision: [YY.Y]%
- Recall: [ZZ.Z]%
- F1-score: [WW.W]%

6.2 Functional Testing

- Verifying that the UI sends input correctly and the API responds without errors.
- Testing different valid and invalid input combinations.
- Checking that the prediction result is displayed correctly to the user.
- Confirming that predictions are appended to prediction_logs.csv when logging is enabled.

7. Challenges & Lessons Learned

During the project we faced several practical challenges:

- Finding a suitable dataset: We had to search for a heart disease dataset that had enough samples, clear labels, and was suitable for machine learning. Not all datasets we found were clean or well-documented.
- Data cleaning and preprocessing: The dataset contained noisy or inconsistent values. We spent time handling missing values, encoding categorical variables and making sure that the preprocessing pipeline used in training is exactly the same during prediction.
- Improving model accuracy: Our initial models did not achieve high accuracy. We tried different algorithms, tuned hyperparameters and experimented with feature selection and scaling to improve the performance while avoiding overfitting.
- Connecting the model with the API and UI: Making sure that the data format sent from the UI matches the format expected by the model required careful debugging and testing.

From these challenges we learned the importance of good data quality, consistent preprocessing, and the need to think about deployment and integration from the beginning, not only about the training accuracy inside the notebook.

8. Conclusion & Future Work

The Heart Disease Prediction System demonstrates how a machine learning model can be integrated into a small end-to-end application. Starting from a raw dataset, we cleaned and analysed the data, trained and evaluated models, and finally deployed the best model behind an API with a simple user interface. The system can assist as a decision-support tool in an educational context.

For future work, the system could be improved by training on larger and more diverse datasets, applying more advanced models and hyperparameter tuning, adding model explainability techniques such as SHAP, and deploying the application on a cloud platform with a proper database and user authentication.