

Heart Disease Risk Prediction System

1. Project Setup

I started by creating Python virtual environment and organizing the project folder structure with directories for data, notebooks, models, deployment files, and reports. This provided a clean workspace to manage all aspects of the project.

2. Data Loading and Exploratory Data Analysis (EDA)

I loaded the heart disease dataset of 5,000 patients, which included 13 input features and a 5-class diagnosis label. I performed full exploratory data analysis to understand feature distributions, identify missing values and outliers, and visualize class distribution across the five diagnosis categories. This helped me understand the dataset comprehensively before preprocessing.

3. Feature Identification and Preprocessing

I **classified the features** into numerical and categorical types:

- **Numerical features:** Age, Blood Pressure, Cholesterol, Max HR, Oldpeak, Number of Affected Vessels, etc.
- **Categorical features:** Sex, Chest Pain Type, Fasting Sugar, ECG, Exercise Angina, ST Slope, Thalassemia Status

For numerical features, I built a preprocessing pipeline to impute missing values (mean or median) and scale values using StandardScaler. For categorical features, I imputed missing values using the most frequent strategy and applied OneHotEncoder to handle unknown categories. I then combined both pipelines using a ColumnTransformer.

4. Train-Test Split and Verification

I split the dataset into 80% training and 20% testing using stratification to maintain class balance. I verified the split by displaying the class distributions of the original dataset, training set, and

testing set in a table showing percentages. After applying the preprocessing pipelines, I checked that the transformed datasets contained zero missing values and that all features were numeric.

5. Model Training and Hyperparameter Tuning

I trained multiple models, including MLP/ANN, Random Forest, SVM, kNN, and Gradient Boosting. For each model, I built a pipeline combining preprocessing and classifier. I performed hyperparameter tuning using GridSearchCV, recording the best parameters, cross-validation accuracy, training accuracy, test accuracy, and training time for every model.

6. Model Comparison and Selection

I created a comparison table summarizing each model's performance, including best CV accuracy, train/test accuracy, overfitting gap, and status (overfit, underfit, best-fit). Sorting the table by test accuracy helped me identify the top-performing model, which I selected for detailed evaluation.

7. Detailed Evaluation of Best Model

I evaluated the best model by:

- Generating a classification report (precision, recall, F1-score) for all five classes
- Computing and visualizing the confusion matrix
- Performing per-class analysis to identify classes with the highest precision and lowest recall, providing clinical interpretation for each
- Displaying feature importance for models that supported it, highlighting the most influential patient indicators

8. Model Persistence

I saved the best-performing model, including the full preprocessing pipeline, feature names (feature_columns.txt), and class names (class_names.txt). I verified the saved model by making predictions on random test samples and custom patient inputs, confirming predictions and class probabilities matched expectations.

9. Flask API Deployment

I deployed the model using Flask, creating a /api/predict endpoint that accepts JSON inputs of 13 patient features and returns the predicted class and per-class probabilities. I ensured the API handled input validation and errors effectively.

10. Frontend Implementation

I developed a responsive HTML frontend (index_25rp20399.html) allowing medical staff to enter patient data and view predictions. The predicted class and color-coded probability bars reflected risk levels:

- Immediate Danger → Red
- Severe → Yellow
- Mild → Orange
- Very Mild → Green
- No Disease → Blue

The design was fully responsive for mobile, tablet, and desktop devices.

USER INTERFACE:

The screenshot displays the 'HEART DISEASE RISK PREDICTOR SYSTEM' interface. It is organized into two main sections: 'Patient Information' and 'Clinical Measurements'. The 'Patient Information' section includes fields for 'Age (years)' (with a numeric input showing 'e.g., 45') and 'Sex' (a dropdown menu showing 'Select...'). The 'Clinical Measurements' section contains several fields: 'Chest Pain Type' (dropdown, 'Select...'), 'Resting Blood Pressure (mm Hg)' (numeric input, 'e.g., 120'), 'Serum Cholesterol (mg/dl)' (numeric input, 'e.g., 200'), 'Fasting Blood Sugar > 120 mg/dl' (dropdown, 'Select...'), 'Resting ECG Results' (dropdown, 'Select...'), 'Maximum Heart Rate Achieved' (numeric input, 'e.g., 150'), 'Exercise Induced Angina' (dropdown, 'Select...'), 'ST Depression (Oldpeak)' (numeric input, 'e.g., 1.0'), 'ST Segment Slope' (dropdown, 'Select...'), 'Major Vessels Colored (0-3)' (numeric input, '0-3'), and 'Thalassemia Status' (dropdown, 'Select...'). At the bottom of the form is a prominent blue button labeled 'ANALYZE PATIENT RISK'.

Heart Disease Risk Predictor System has a clean and intuitive interface that organizes patient information and clinical measurements in clear sections. Users can easily enter data using dropdowns and numeric fields and get instant, color-coded diagnostic results with risk probabilities. The interface is designed to be simple and straightforward, making it easy to understand and use.

11. Testing, Documentation, and Video Explanation

I thoroughly tested the system for API-model consistency, input validation, error handling, and responsiveness across devices. I documented all procedures, results, and instructions in a README file and a project report (PDF), including screenshots and explanations of model performance and frontend functionality. Additionally, I recorded a video explaining the system, demonstrating how the model works, how to use the frontend, and how the predictions and probability outputs are interpreted by medical staff.