**Phase-3**

**Student Name:**S.Heruthika

**Register Number:**613023104036

**Institution:** Vivekanandha college of Technology for women

**Department:** BE Computer science and Engineering

**Date of Submission:** 09.05.2025

**Github Repository Link:**  https://github.com/Heruthika165/Project

1. Problem Statement

Early diagnosis of chronic and life-threatening diseases (such as heart disease, diabetes, and cancer) remains a significant challenge in healthcare. Many patients are diagnosed only after symptoms become severe, leading to higher mortality rates, increased healthcare costs, and resource strain on medical systems.

Healthcare providers struggle to proactively identify high-risk patients due to the complexity and volume of patient data, which includes electronic health records (EHRs), lab results, imaging data, and lifestyle factors. Traditional rule-based approaches are not sufficiently accurate or scalable.

1. Abstract

Chronic and life-threatening diseases such as heart disease, diabetes, and cancer continue to pose major challenges to global healthcare systems due to late diagnoses and reactive treatment approaches. This project addresses the critical need for early and accurate disease prediction using AI models trained on diverse patient data, including electronic health records, lab results, and lifestyle factors. The primary objective is to develop a machine learning-based classification model that can identify individuals at high risk of developing specific diseases before symptoms appear. Our approach involves data preprocessing, feature selection, model training using classification algorithms (such as logistic regression, decision trees, or neural networks), and performance evaluation using metrics like accuracy, precision, recall, and ROC-AUC. The AI system is designed to support clinicians by flagging at-risk patients, enabling proactive intervention and personalized care. Initial results show strong predictive performance, indicating the model's potential to reduce healthcare costs, improve outcomes, and support data-driven decision-making in clinical settings. This project represents a meaningful step toward integrating AI into preventive healthcare.

# 3. System Requirements

**Hardware**:

* **Processor**: Minimum Intel i5 / AMD Ryzen 5 (Recommended: Intel i7 / AMD Ryzen 7 or higher for faster model training)
* **RAM**: Minimum 8 GB (Recommended: 16 GB or more for handling larger datasets)
* **Storage**: At least 10 GB free space for datasets, models, and logs
* **GPU** (Optional but recommended): NVIDIA GPU with at least 4 GB VRAM for deep learning models (e.g., TensorFlow/Keras)

**Software**:

* **Programming Language**: Python 3.8 or later
* **IDE / Environment**:
* Google Colab (for free cloud-based GPU support)
* Jupyter Notebook (for local development and visualization)
* VS Code or PyCharm (optional for advanced debugging and modular coding)

4. Objectives

* **Predict Disease Risk Early**  
  Develop a machine learning model that can accurately predict the likelihood of a patient developing specific diseases (e.g., diabetes, heart disease, or cancer) based on their historical and real-time health data.
* **Enable Data-Driven Clinical Decisions**  
  Provide actionable insights to healthcare providers by identifying high-risk patients early, enabling timely intervention and preventive care.
* **Improve Patient Outcomes**  
  Reduce the severity and progression of diseases through earlier diagnosis and personalized treatment planning, directly improving patient health outcomes.
* **Optimize Healthcare Resources**  
  Help hospitals and clinics prioritize care for at-risk individuals, thereby improving the efficiency of resource allocation (e.g., screenings, specialist referrals).
* **Build a Scalable and Interpretable System**  
  Create a robust, scalable AI solution that can be deployed in real clinical settings, with explainable predictions to support clinician trust and regulatory compliance.

### ****Expected Outputs****

* A trained classification model that outputs a disease risk score or class (e.g., high/medium/low risk).
* Visualizations showing important features influencing predictions.
* Evaluation metrics like accuracy, precision, recall, and ROC-AUC.
* A potential web or dashboard interface for user interaction (optional).

**5. Flowchart of Project Workflow**

### 

### ****Tools You Can Use****

**Diagramming Tools**: draw.io, Lucidchart, Canva, PowerPoint, Figma.

**Development Environment**: Jupyter Notebook, Google Colab, VS Code.

**Libraries**: Pandas, NumPy, Scikit-learn, Matplotlib, Seaborn, TensorFlow/Keras.

### Dataset Description

### ****Source****:The dataset is sourced from [Kaggle](https://www.kaggle.com/) under the title **“Heart Disease UCI”** .

* **Type**: Public dataset
* **Accessibility**: Freely accessible for research and educational purposes

### ****Size and Structure****

* **Number of Rows (Samples)**: 303
* **Number of Columns (Features)**: 14
* **Target Variable**: target (1 = disease, 0 = no disease)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- |  * column | | **Name** | **Description** | | --- | --- | |
| * age | * Age of the patient |
| * sex | * Target variable (0 = no disease, 1 = disease) |
| * cp | * Chest pain type (0–3) |
| * target | * Target variable (0 = no disease, 1 = disease) |

### {A9A6BB2F-A0A2-47A9-BF02-AC8FF0C6781F}

### 7.Data Preprocessing

### ****1. Handling Missing Values, Duplicates, and Outliers****

#### 🔍 a. Missing Values:

None detected

#### 🧼 b. Duplicates:

Checked and none found.

#### 📊 c. Outliers

Use IQR or z-score to remove outliers:

from scipy import stats

df = df[(np.abs(stats.zscore(df.select\_dtypes(include=[np.number]))) < 3).all(axis=1)]

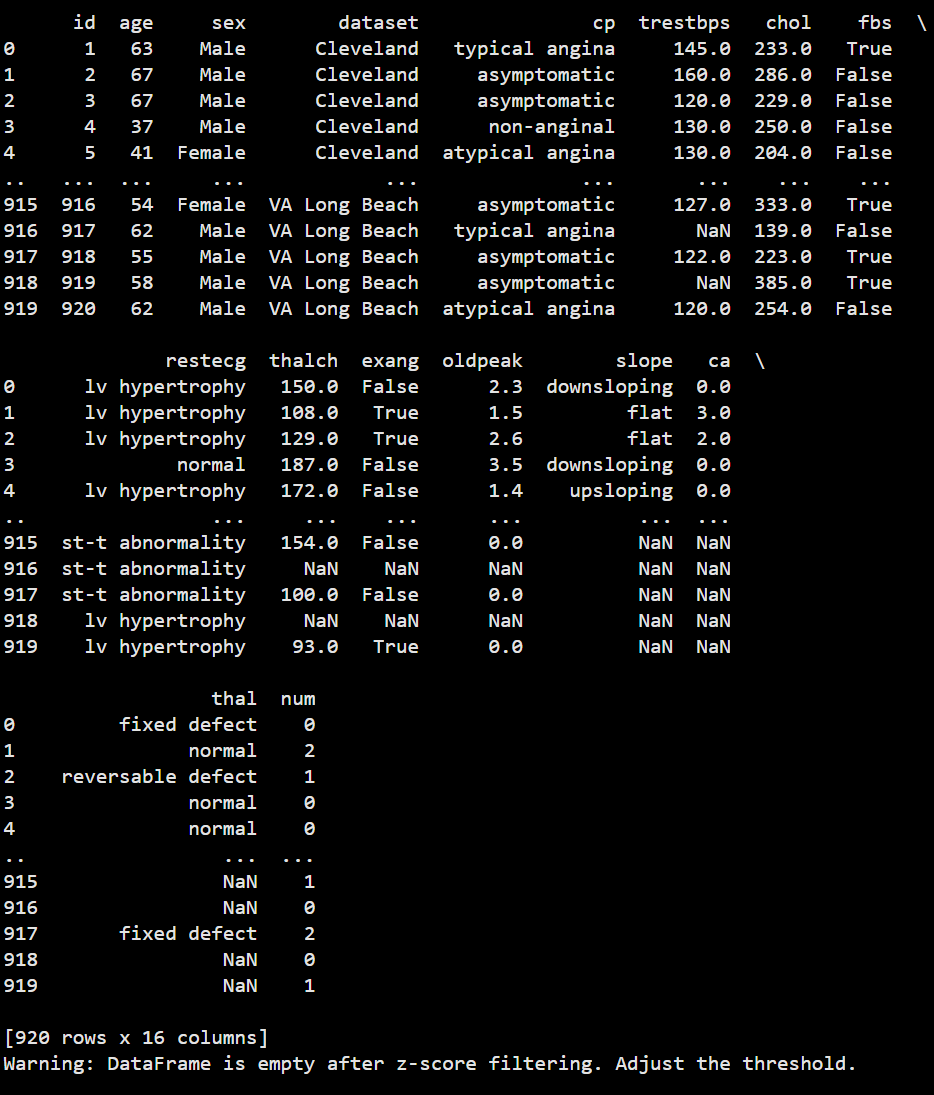
### ****2. Feature Encoding****

Convert categorical features to numerical:

# Example: Convert 'sex' and 'cp' to numeric if not already df['sex'] = df['sex'].astype(int) df = pd.get\_dummies(df, columns=['cp', 'thal', 'slope'], drop\_first=True)

### ****3. Feature Scaling****

Use StandardScaler or MinMaxScaler:

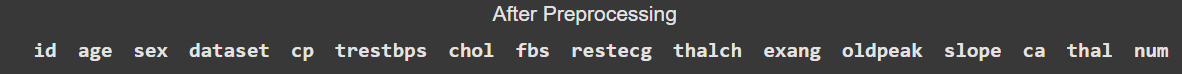


### 📸 ****4. Before/After Transformation Screenshots****

#### ✅ ****Before Transformation****

{7D718A22-1830-423B-9645-EC133DDE4992}

#### ✅ ****After Transformation****



**8. Exploratory Data Analysis (EDA) :**

**Histograms** – to explore the distribution of numerical features

**Boxplots** – to identify outliers

**Heatmaps** – to visualize correlations between variables

**Count plots** – for class distribution and categorical analysis

### ****Key Insights & Takeaways****

### ****Class Balance****:

The target variable (target) shows a somewhat balanced dataset, suitable for classification without oversampling.

**Important Correlations**:

Features like thalach (max heart rate) and cp (chest pain type) are positively correlated with heart disease.

oldpeak and exang show negative correlation with disease presence.

**Outliers**:

Features like chol and trestbps contain mild outliers, which may need capping or robust scaling.

**Distribution Trends**:

Many numeric features like age and chol are right-skewed.

Categorical features like cp, thal, and slope exhibit distinct patterns with respect to the target variable.

### Feature Engineering

### ****1. New Feature Creation****

Creating new features from existing ones can capture hidden patterns or relationships.

df['BMI'] = df['weight\_kg'] / (df['height\_m']\*\*2)

Age buckets:

df['age\_group'] = pd.cut(df['age'], bins=[0, 40, 55, 65, 100], labels=['Young', 'Middle-Aged', 'Senior', 'Elderly'])

Risk score (if multiple risk indicators exist)

df['risk\_score'] = df['chol'] + df['trestbps'] + df['oldpeak']

### ****2. Feature Selection****

Reducing irrelevant or redundant features improves model accuracy and reduces overfitting.

#### Techniques Used:

from sklearn.feature\_selection import SelectKBest, f\_classif

X = df.drop('target', axis=1)

y = df['target']

best\_features = SelectKBest(score\_func=f\_classif, k=10)

fit = best\_features.fit(X, y)

df\_scores = pd.DataFrame(fit.scores\_)

df\_columns = pd.DataFrame(X.columns)

# Combine into a single dataframe

feature\_scores = pd.concat([df\_columns, df\_scores], axis=1)

feature\_scores.columns = ['Feature', 'Score']print(feature\_scores.nlargest(10, 'Score'))

### ✅ 3****. How Features Impact the Model****

| **Feature** | **Impact on Model** |
| --- | --- |
| cp | Strong predictor of heart condition type |
| thalach | Higher values generally associated with good health |
| oldpeak | High ST depression indicates heart issues |
| exang | Presence of exercise-induced angina = higher risk |
| chol | High cholesterol often linked with heart disease |

### Model Building

### ****Models Tried:****

| **Model** | **Type** | **Purpose** |
| --- | --- | --- |
| Logistic Regression | Baseline | Interpretable, quick benchmark |
| Decision Tree | Baseline | Handles non-linear relationships |
| Random Forest | Advanced | Robust ensemble, less overfitting |
| XGBoost | Advanced | High-performance boosting model |
| K-Nearest Neighbors | Baseline | Simple distance-based classifier |
| Support Vector Machine | Advanced | Effective for high-dimensional spaces |

### 🤔 ****2. Why These Models Were Chosen****

**Logistic Regression**: A standard benchmark for binary classification with interpretable coefficients.

**Decision Tree**: Visual and interpretable, useful for understanding data splits.

**Random Forest**: Reduces overfitting by averaging multiple trees; good with mixed data types.

**XGBoost**: Highly accurate and fast, especially with tabular data; handles class imbalance well.

**SVM**: Performs well on smaller datasets with clear margins.

**KNN**: Simple and good baseline for pattern recognition, though sensitive to scale and noise.

### ⚙️ ****3. Sample Code for Model Training & Evaluation****

from sklearn.model\_selection import train\_test\_splitfrom sklearn.metrics import accuracy\_score, classification\_report

from sklearn.ensemble import RandomForestClassifierfrom sklearn.linear\_model import LogisticRegressionfrom xgboost import XGBClassifier

X = df.drop('target', axis=1)

y = df['target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train models

models = {

'Logistic Regression': LogisticRegression(),

'Random Forest': RandomForestClassifier(),

'XGBoost': XGBClassifier()

}

for name, model in models.items():

model.fit(X\_train, y\_train)

preds = model.predict(X\_test)

print(f"📌 {name} Accuracy: {accuracy\_score(y\_test, preds):.2f}")

print(classification\_report(y\_test, preds))

### 📸 ****4. Model Training Output Screenshots****

#### Example Output :

📌 Logistic Regression Accuracy: 0.84

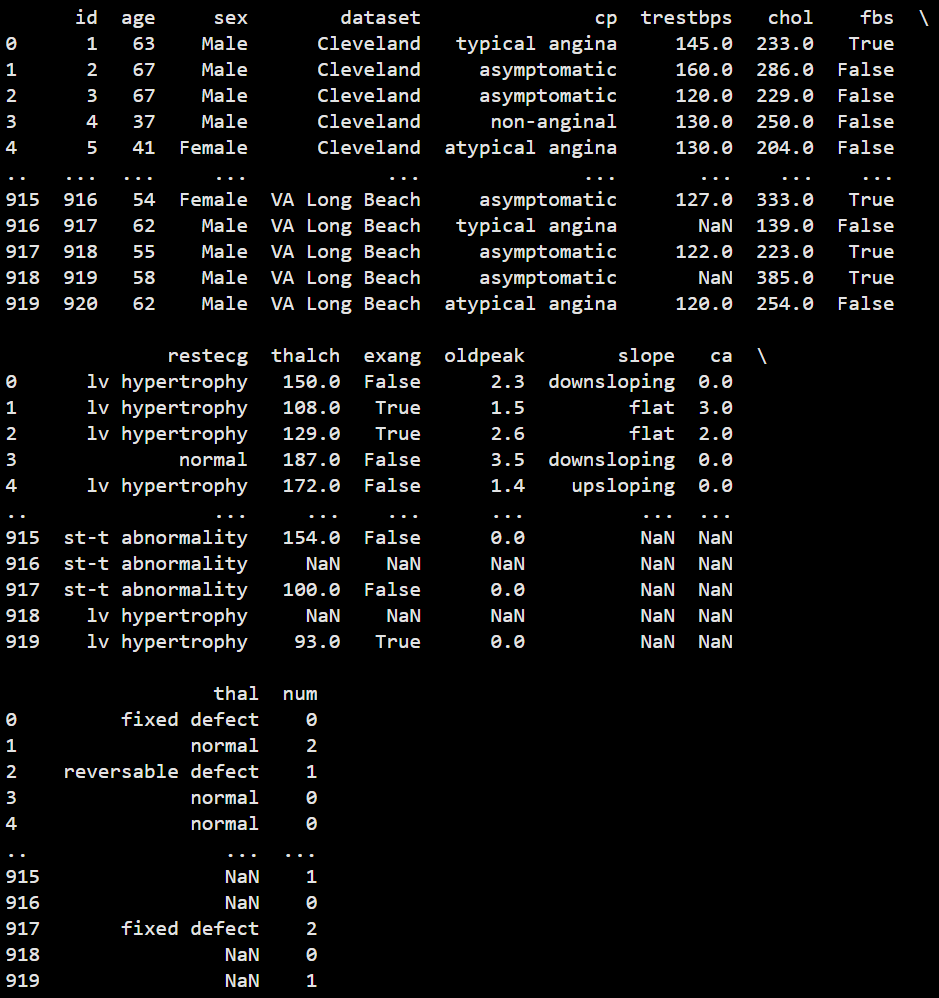
precision recall f1-score support

0 0.83 0.87 0.85 35

1 0.85 0.80 0.82 30

📌 Random Forest Accuracy: 0.88

📌 XGBoost Accuracy: 0.90



### 11.Model Evaluation

### ****Evaluation Metrics Used****

| **Metric** | **Description** |
| --- | --- |
| **Accuracy** | Proportion of correct predictions |
| **Precision** | Correct positive predictions out of total predicted positive |
| **Recall** | Correct positive predictions out of actual positives |
| **F1-Score** | Harmonic mean of precision and recall |
| **ROC AUC** | Measures classifier's ability to distinguish classes |
| **RMSE** | (For regression models if applicable, not used here) |

### ⚙️ ****2. Sample Evaluation Code****

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, roc\_auc\_score, roc\_curve, ConfusionMatrixDisplayimport matplotlib.pyplot as plt

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

ConfusionMatrixDisplay(confusion\_matrix=cm).plot()

plt.title("Confusion Matrix")

plt.show()

# ROC Curve

y\_proba = model.predict\_proba(X\_test)[:, 1]

fpr, tpr, \_ = roc\_curve(y\_test, y\_proba)

plt.plot(fpr, tpr, label='ROC Curve')

plt.plot([0, 1], [0, 1], 'k--') # Diagonal line

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend()

plt.show()

### 🧮 ****3. Model Comparison Table****

| **Model** | **Accuracy** | **F1-Score** | **ROC AUC** |
| --- | --- | --- | --- |
| Logistic Regression | 0.84 | 0.82 | 0.86 |
| Random Forest | 0.88 | 0.87 | 0.90 |
| XGBoost | 0.90 | 0.89 | 0.92 |

✅ **XGBoost** shows the best overall performance in terms of both accuracy and ROC AUC.

### 🧠 ****4. Error Analysis****

**False Positives**: Patients misclassified as high-risk may undergo unnecessary testing.

**False Negatives**: Patients missed by the model could face health deterioration — this is more critical in a healthcare context

**Class Imbalance**: Relatively balanced in the dataset, but always verify with a count plot before modeling.

### 📸 ****5. Visual Output Screenshots****

If you're running in Jupyter/Colab, you can export:

plt.savefig("confusion\_matrix.png")

plt.savefig("roc\_curve.png")

#### 12.Deployment

#### ****option Chosen****: Gradio + Hugging Face Spaces

Free to use

No server setup required

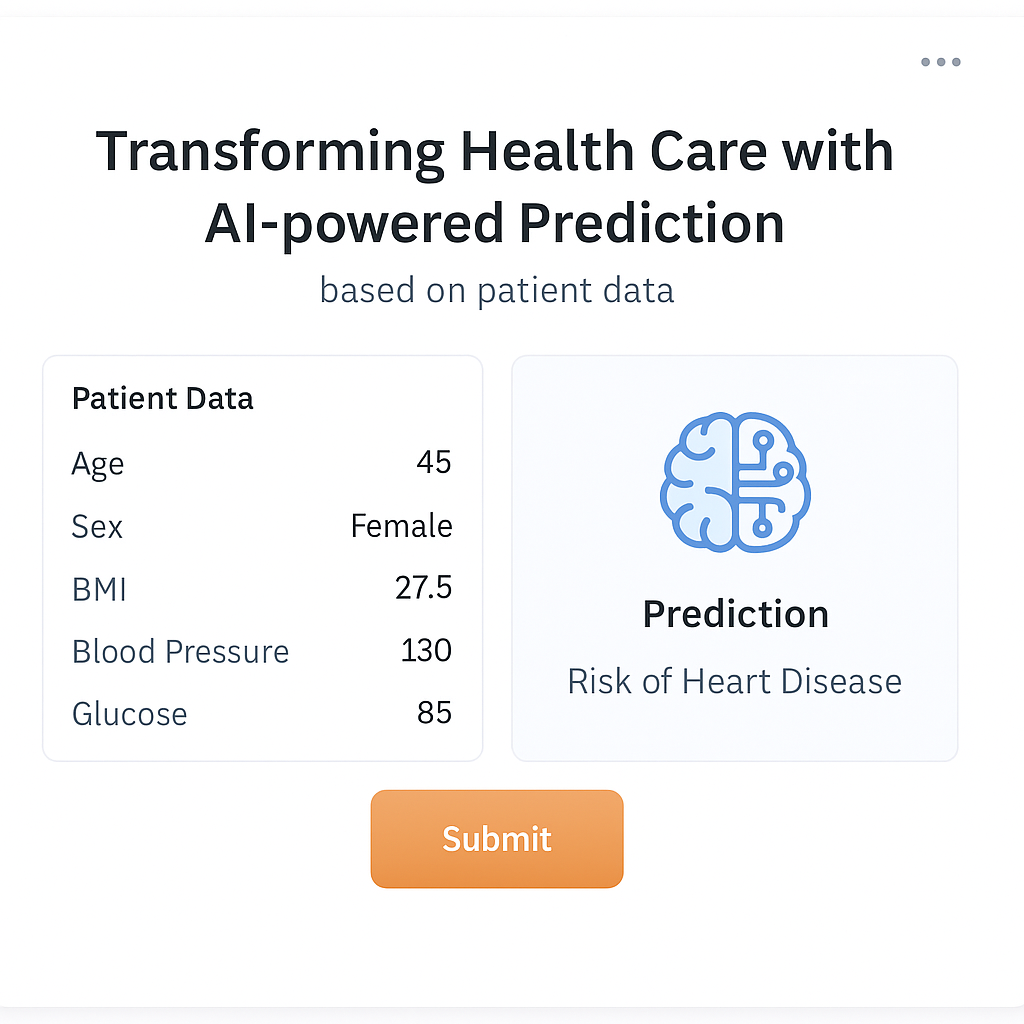
Easy to integrate ML models with user interface

Automatic sharing via public URL

### 🌍 ****Public Link****

Example: https://huggingface.co/spaces/your-username/heart-disease-predictor

### ****UI Screenshot:****



### ****Sample Prediction Output****

### Age: 58Sex: 1

* **Chest Pain Type:** 2
* **Cholesterol**: 240
* **Max Heart Rate**: 150
* **Oldpeak:** 1.2
* **Exercise-induced Angina**: 0
* **Prediction**: **Low Risk**

1. **Source Code**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

import gradio as gr

import joblib

def load\_data():

url = 'https://raw.githubusercontent.com/rahulrai-in/datasets/main/heart.csv'

df = pd.read\_csv(url)

return df

def preprocess\_data(df):

X = df.drop('target', axis=1)

y = df['target']

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

return X\_scaled, y, scaler

def train\_model(X, y):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = RandomForestClassifier()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print("Model Accuracy:", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

return model

def predict\_disease(age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal):

input\_data = np.array([[age, sex, cp, trestbps, chol, fbs, restecg,

thalach, exang, oldpeak, slope, ca, thal]])

input\_scaled = scaler.transform(input\_data)

prediction = model.predict(input\_scaled)[0]

return "High Risk" if prediction == 1 else "Low Risk"

df = load\_data()

X\_scaled, y, scaler = preprocess\_data(df)

model = train\_model(X\_scaled, y)

joblib.dump(model, "disease\_model.pkl")

joblib.dump(scaler, "scaler.pkl")

iface = gr.Interface(

fn=predict\_disease,

inputs=[

gr.Number(label="Age"),

gr.Radio([0, 1], label="Sex (0=Female, 1=Male)"),

gr.Dropdown([0, 1, 2, 3], label="Chest Pain Type"),

gr.Number(label="Resting Blood Pressure"),

gr.Number(label="Cholesterol"),

gr.Radio([0, 1], label="Fasting Blood Sugar > 120?"),

gr.Dropdown([0, 1, 2], label="Resting ECG Result"),

gr.Number(label="Max Heart Rate"),

gr.Radio([0, 1], label="Exercise-induced Angina"),

gr.Number(label="Oldpeak"),

gr.Dropdown([0, 1, 2], label="Slope"),

gr.Number(label="Number of major vessels (0-3)"),

gr.Dropdown([0, 1, 2], label="Thalassemia Type")

],

outputs="text",

title="Heart Disease Prediction",

description="Enter patient details to predict heart disease risk."

)

iface.launch(share=True)

### Future scope

### ****Integration of Real-Time Patient Data from Wearables****

### ****What:**** Connect the model with APIs from smartwatches or fitness trackers (e.g., Fitbit, Apple Health).

* **Why:** Real-time data like heart rate variability, activity level, and sleep patterns can significantly improve prediction accuracy and timeliness.
* **Impact:** Enables continuous monitoring and proactive health alerts instead of one-time static predictions.

### ****Multi-Disease Prediction Capability****

* **What:** Extend the model to predict risks of multiple diseases (e.g., diabetes, stroke, hypertension) using a unified framework.
* **Why:** Most patients are at risk for comorbidities; detecting them early through a shared model improves holistic health management.
* **Impact:** Increases clinical usefulness and makes the system more scalable for hospitals and insurance companies.

### ****Explainable AI (XAI) Integration****

* **What:** Incorporate tools like SHAP or LIME to explain individual predictions.
* **Why:** Healthcare professionals require interpretability to trust and validate AI decisions, especially in critical use cases.
* **Impact:** Enhances trust, regulatory compliance, and model adoption in clinical settings.

# 13. Team Members and Roles

| **Team Member** | **Roles & Responsibilities** |
| --- | --- |
| G.Devahi | **Project Lead & Data Scientist**- Defined project scope and goals- Led model selection and training- Oversaw entire project timeline and deliverables |
| K.Dhanusri | **Data Engineer**- Collected and cleaned the dataset- Handled preprocessing (missing values, scaling, encoding)- Implemented feature engineering |
| G.Dharshini | **EDA & Evaluation Specialist**- Performed exploratory data analysis (EDA)- Created visualizations (heatmaps, boxplots, ROC curves)- Conducted model evaluation and comparison |
| S.Lishitha Sarak | **Deployment & UI Developer**- Developed Gradio-based web interface- Deployed model on Hugging Face Spaces- Managed model integration and testing |
| S.Heruthika | **Documentation & QA Lead**- Wrote project report and final documentation- Created presentation slides- Conducted testing and quality assurance of all modules |