**PROJECT CODE DOCUMENTATION**

1. **Software and Libraries Used**

This project relies on a combination of machine learning, web development, and AI integration libraries. Below is a comprehensive list of the technologies used:

**Programming Language**

* Python 3.8+

**Libraries and Frameworks**

* streamlit: for building the interactive web application
* pandas, numpy: for data handling and numerical operations
* scikit-learn: for building and evaluating machine learning models (Voting Classifier, Logistic Regression, Random Forest, Support Vector Machine)
* joblib: for saving and loading the trained model
* matplotlib, seaborn: for data visualization (optional if used)
* openai: to integrate OpenAI GPT-based conversational assistant
* google.generativeai: to access Google Gemini API for the AI Chemist
* PIL (Pillow): for handling image uploads (prescriptions)
* fpdf: to optionally export results as PDF reports
* os, io, datetime: for file operations and timestamp management

1. **Instructions to Execute the Code**

Follow these steps to properly run the project:

#### ****2.1 Installation Requirements****

Make sure the following libraries are installed. You can run:

pip install streamlit pandas numpy scikit-learn joblib openai google-generativeai pillow fpdf

#### ****2.2 Prepare API Keys****

Create a .env file or set environment variables for:

OPENAI\_API\_KEY = your\_openai\_key\_here

GOOGLE\_API\_KEY = your\_google\_gemini\_key\_here

Alternatively, you can directly assign the API keys in the script if working locally (not recommended for deployment due to security).

#### ****2.3 Running the Application****

Navigate to the folder containing your Python script and execute:

streamlit run your\_script\_name.py

Make sure the following files are in the same directory:

* heart.csv: Dataset used to train the model
* ensemble\_model.pkl: Pre-trained model file (will be auto-generated if not present)

Streamlit will automatically launch the web app in your browser.

### **Description of the Code**

* The script begins by checking if a pre-trained model exists. If not, it loads the heart.csv dataset, trains an ensemble classifier (VotingClassifier with LogisticRegression, RandomForest, and SVM), and saves the model using joblib.
* The application is built with Streamlit and allows users to:
  + Input clinical data
  + Predict CAD risk
  + Upload images for the AI Chemist to interpret prescriptions
  + Query an OpenAI GPT-powered chatbot
* The Google Gemini API is used to handle both image and text queries for medication guidance.
* Results can optionally be exported to PDF using the fpdf module.

4. Code

import streamlit as st

import joblib

import pandas as pd

import openai

import os

import google.generativeai as genai

from sklearn.ensemble import VotingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.preprocessing import StandardScaler

from PIL import Image

from fpdf import FPDF

import datetime

import io

import numpy as np

# Load API keys securely

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

GOOGLE\_API\_KEY = os.getenv("GOOGLE\_API\_KEY")

genai.configure(api\_key=GOOGLE\_API\_KEY)

# Load and preprocess dataset

if not os.path.exists("ensemble\_model.pkl"):

data = pd.read\_csv("heart.csv")

X = data.drop(columns=['target'])

y = data['target']

# Standardize features

scaler = StandardScaler()

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

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

# Define models with balanced class weights

model1 = LogisticRegression(class\_weight='balanced', max\_iter=1000)

model2 = RandomForestClassifier(class\_weight='balanced', n\_estimators=100)

model3 = SVC(probability=True, class\_weight='balanced')

# Create Voting Classifier

voting\_clf = VotingClassifier(estimators=[

('lr', model1),

('rf', model2),

('svm', model3)

], voting='soft')

# Cross-validation

scores = cross\_val\_score(voting\_clf, X\_scaled, y, cv=5)

print(f"Cross-validation accuracy: {scores.mean():.2f} (+/- {scores.std():.2f})")

# Train and save ensemble model

voting\_clf.fit(X\_scaled, y)

joblib.dump(voting\_clf, "ensemble\_model.pkl")

else:

data = pd.read\_csv("heart.csv")

X = data.drop(columns=['target'])

scaler = joblib.load("scaler.pkl")

# Load the trained ensemble model

model = joblib.load("ensemble\_model.pkl")

def clean\_text(text):

"""Convert special characters to ASCII equivalents, handling numeric inputs"""

if text is None:

return ""

if not isinstance(text, str):

text = str(text)

replacements = {

'≤': '<=',

'≥': '>=',

'–': '-',

'—': '-',

'‘': "'",

'’': "'",

'“': '"',

'”': '"',

'…': '...'

}

for k, v in replacements.items():

text = text.replace(k, v)

return text

def explain\_prediction(prediction, input\_data):

explanation = []

input\_data = input\_data.iloc[0] # Convert DataFrame row to Series

if prediction == 1:

explanation.append("🚨 \*\*High Risk Factors Detected:\*\*")

if input\_data['age'] > 50:

explanation.append(f"- Age ({input\_data['age']}) increases risk")

if input\_data['chol'] > 240:

explanation.append(f"- High cholesterol ({input\_data['chol']} mg/dL)")

if input\_data['trestbps'] > 140:

explanation.append(f"- Elevated blood pressure ({input\_data['trestbps']} mmHg)")

if input\_data['exang'] == 1:

explanation.append("- Exercise-induced angina is concerning")

if input\_data['oldpeak'] > 2:

explanation.append(f"- Significant ST depression ({input\_data['oldpeak']} mm)")

if input\_data['ca'] > 0:

explanation.append(f"- {4- input\_data['ca']} major vessels showing blockage")

else:

explanation.append("✅ \*\*Protective Factors:\*\*")

if input\_data['age'] <= 50:

explanation.append(f"- Younger age ({input\_data['age']}) reduces risk")

if input\_data['chol'] <= 200:

explanation.append(f"- Healthy cholesterol level ({input\_data['chol']} mg/dL)")

if input\_data['trestbps'] <= 120:

explanation.append(f"- Normal blood pressure ({input\_data['trestbps']} mmHg)")

return "\n".join(explanation)

def generate\_lifestyle\_recommendations(input\_data, prediction):

recommendations = []

# Enhanced General Tips

recommendations.append("- 🏃‍♂️ \*\*Exercise Regularly\*\*: 30-45 minutes of brisk walking, 5 days/week")

recommendations.append("- 🥗 \*\*Balanced Diet\*\*: Focus on whole grains, lean proteins, and colorful vegetables")

recommendations.append("- 🫒 \*\*Limit Oil Intake\*\*: Use <3 tsp oil per day, prefer olive/canola oil")

recommendations.append("- 🧂 \*\*Reduce Salt\*\*: Aim for <5g (1 tsp) salt daily including hidden salts")

recommendations.append("- 💧 \*\*Hydration\*\*: Drink 2-3 liters of fluids daily (water, herbal teas)")

recommendations.append("- 🌾 \*\*Increase Fiber\*\*: Gradually add whole grains, fruits with skin, legumes")

recommendations.append("- 🚭 \*\*Tobacco Free\*\*: Complete avoidance of smoking/chewing tobacco")

recommendations.append("- 🍷 \*\*Alcohol Moderation\*\*: ≤1 drink/day for women, ≤2 for men (1 drink = 14g alcohol)")

recommendations.append("- 😴 \*\*Sleep Hygiene\*\*: 7-9 hours quality sleep, maintain consistent schedule")

recommendations.append("- 🧘 \*\*Stress Management\*\*: Daily 10-15 min meditation/deep breathing")

# Personalized recommendations

if input\_data['age'].values[0] > 50:

recommendations.append("\n### For Age 50+:")

recommendations.append("- 📅 \*\*Annual Checkups\*\*: Full lipid profile + cardiac evaluation")

recommendations.append("- 💊 \*\*Aspirin Consideration\*\*: Discuss low-dose aspirin with your doctor")

recommendations.append("- 🏋️ \*\*Strength Training\*\*: Add 2 days/week resistance exercises")

if input\_data['chol'].values[0] > 240:

recommendations.append("\n### Cholesterol Management:")

recommendations.append("- 🥑 \*\*Healthy Fats\*\*: Increase nuts, seeds, fatty fish (salmon)")

recommendations.append("- 🚫 \*\*Avoid Trans Fats\*\*: Check labels for 'partially hydrogenated oils'")

recommendations.append("- 🌾 \*\*Soluble Fiber\*\*: 10-25g/day from oats, psyllium, apples")

if prediction == 1:

recommendations.append("\n### Medical Priority Actions:")

recommendations.append("- 🩺 \*\*Doctor Consultation\*\*: Within 2 weeks for cardiac evaluation")

recommendations.append("- 💊 \*\*Medication Review\*\*: Statins/antihypertensives if indicated")

recommendations.append("- 📊 \*\*Monitoring Plan\*\*: Weekly BP + monthly lipid checks initially")

recommendations.append("- 🚨 \*\*Emergency Signs\*\*: Know symptoms requiring immediate care:")

recommendations.append(" - Chest pressure/pain lasting >15 minutes")

recommendations.append(" - Sudden shortness of breath with sweating")

return recommendations

def predict\_heart\_disease(input\_data):

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

return model.predict(input\_scaled)[0] # Only return prediction (0 or 1)

def preprocess\_input(age, gender, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal):

gender = 1 if gender == "Male" else 0

fbs = 1 if fbs == "True" else 0

exang = 1 if exang == "Yes" else 0

cp\_dict = {"Typical Angina": 0, "Atypical Angina": 1, "Non-anginal Pain": 2, "Asymptomatic": 3}

cp = cp\_dict[cp]

restecg\_dict = {"Normal": 0, "ST-T wave abnormality": 1, "Left ventricular hypertrophy": 2}

restecg = restecg\_dict[restecg]

slope\_dict = {"Upsloping": 0, "Flat": 1, "Downsloping": 2}

slope = slope\_dict[slope]

thal\_dict = {"Normal": 0, "Fixed Defect": 1, "Reversible Defect": 2}

thal = thal\_dict[thal]

return pd.DataFrame({

'age': [age], 'gender': [gender], 'cp': [cp], 'trestbps': [trestbps],

'chol': [chol], 'fbs': [fbs], 'restecg': [restecg], 'thalach': [thalach],

'exang': [exang], 'oldpeak': [oldpeak], 'slope': [slope], 'ca': [ca],

'thal': [thal]

})

def generate\_pdf(input\_data, prediction, recommendations):

pdf = FPDF()

pdf.add\_page()

# Set font - using Arial as it's widely available

pdf.set\_font("Arial", size=12)

# Title

pdf.set\_font("Arial", 'B', 16)

pdf.cell(200, 10, txt="Heart Disease Risk Report", ln=True, align='C')

pdf.ln(10)

# Date

pdf.set\_font("Arial", size=10)

pdf.cell(0, 10, txt=f"Generated on: {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}", ln=True)

pdf.ln(10)

# Patient Information

pdf.set\_font("Arial", 'B', 12)

pdf.cell(0, 10, txt="Patient Information:", ln=True)

pdf.set\_font("Arial", size=10)

readable\_data = {

'Age': str(input\_data['age'][0]),

'Gender': 'Male' if input\_data['gender'][0] == 1 else 'Female',

'Resting Blood Pressure': f"{input\_data['trestbps'][0]} mmHg",

'Cholesterol': f"{input\_data['chol'][0]} mg/dL",

'Fasting Blood Sugar': '>120 mg/dL' if input\_data['fbs'][0] == 1 else '<=120 mg/dL',

'Max Heart Rate': str(input\_data['thalach'][0]),

'ST Depression': str(input\_data['oldpeak'][0])

}

for key, value in readable\_data.items():

pdf.cell(0, 10, txt=f"{key}: {value}", ln=True)

pdf.ln(10)

# Risk Assessment

pdf.set\_font("Arial", 'B', 12)

pdf.cell(0, 10, txt="Risk Assessment:", ln=True)

pdf.set\_font("Arial", size=10)

risk\_text = "HIGH RISK of heart disease" if prediction == 1 else "LOW RISK of heart disease"

pdf.cell(0, 10, txt=f"Assessment: {risk\_text}", ln=True)

pdf.ln(10)

# Recommendations

pdf.set\_font("Arial", 'B', 12)

pdf.cell(0, 10, txt="Lifestyle Recommendations:", ln=True)

pdf.set\_font("Arial", size=10)

for rec in recommendations:

clean\_text = rec.replace("###", "").replace("\*\*", "")

clean\_text = clean\_text.replace("🫒", "[Oil]").replace("🧂", "[Salt]")

clean\_text = clean\_text.replace("💧", "[Water]").replace("🌾", "[Fiber]")

clean\_text = clean\_text.replace("🏃‍♂️", "[Exercise]").replace("🥗", "[Diet]")

clean\_text = clean\_text.replace("🚭", "[Smoking]").replace("🍷", "[Alcohol]")

clean\_text = clean\_text.replace("😴", "[Sleep]").replace("🧘", "[Stress]")

clean\_text = clean\_text.replace("📅", "[Checkups]").replace("💊", "[Medication]")

clean\_text = clean\_text.replace("🏋️", "[Strength]").replace("🥑", "[Healthy Fats]")

clean\_text = clean\_text.replace("🚫", "[Avoid]").replace("🩺", "[Doctor]")

clean\_text = clean\_text.replace("📊", "[Monitoring]").replace("🚨", "[Emergency]")

clean\_text = ''.join(char for char in clean\_text if ord(char) < 128)

pdf.multi\_cell(0, 10, txt=clean\_text)

return pdf.output(dest='S').encode('latin-1')

def main():

st.title('🫀 AI-Powered Coronary Artery Disease (CAD) Risk Assessment')

st.markdown("""

<style>

.description-box {

background-color: #092a66;

border-radius: 10px;

padding: 15px;

margin-bottom: 20px;

}

.risk-high {

color: #ff4b4b;

font-weight: bold;

}

.risk-low {

color: #006400;

font-weight: bold;

}

.section-title {

color: #2c3e50;

border-bottom: 2px solid #2c3e50;

padding-bottom: 5px;

}

</style>

""", unsafe\_allow\_html=True)

option = st.sidebar.radio("Choose an Application:", ["Heart Disease Risk Assessment", "AI Chemist"])

if option == "Heart Disease Risk Assessment":

st.header("🩺 Heart Disease Risk Assessment")

st.markdown("""

<div class="description-box">

This tool assesses your risk of coronary artery disease based on key health indicators.

Please provide accurate information for the most reliable assessment.

</div>

""", unsafe\_allow\_html=True)

with st.expander("🔍 About This Assessment"):

st.write("""

This risk assessment uses machine learning to analyze multiple factors that contribute to heart disease risk.

The model combines three different algorithms (Logistic Regression, Random Forest, and SVM) for more accurate predictions.

""")

col1, col2 = st.columns(2)

with col1:

age = st.slider('Age', 29, 77, 50)

st.markdown("""

<div class="description-box">

<b>ℹ️ Age Factor:</b> Risk increases with age. Men over 45 and women over 55 are at higher risk.

</div>

""", unsafe\_allow\_html=True)

gender\_options = {

"Male": "Men generally develop heart disease earlier than women.",

"Female": "Women's risk increases after menopause, and symptoms may differ."

}

gender = st.selectbox("Gender", list(gender\_options.keys()))

st.markdown(f"""

<div class="description-box">

<b>ℹ️ Gender Difference:</b> {gender\_options[gender]}

</div>

""", unsafe\_allow\_html=True)

chest\_pain\_desc = {

"Typical Angina": "Predictable chest pain during exertion, relieved by rest - classic sign of reduced blood flow to heart.",

"Atypical Angina": "Less predictable chest discomfort that may not follow typical patterns.",

"Non-anginal Pain": "Chest discomfort unlikely to be heart-related (e.g., digestive or muscular).",

"Asymptomatic": "No chest pain (silent ischemia can still indicate heart disease)."

}

cp = st.selectbox("Chest Pain Type", list(chest\_pain\_desc.keys()))

st.markdown(f"""

<div class="description-box">

<b>ℹ️ Chest Pain Info:</b> {chest\_pain\_desc[cp]}

</div>

""", unsafe\_allow\_html=True)

trestbps = st.slider('Resting Blood Pressure (mm Hg)', 94, 200, 120)

st.markdown(f"""

<div class="description-box">

<b>ℹ️ Blood Pressure Guide:</b>

- Normal: Below 120/80 mmHg

- Elevated: 120-129/<80 mmHg

- High: 130+/80+ mmHg

Your input: {trestbps} mmHg (systolic)

</div>

""", unsafe\_allow\_html=True)

chol = st.slider('Cholesterol (mg/dl)', 126, 564, 200)

st.markdown(f"""

<div class="description-box">

<b>ℹ️ Cholesterol Levels:</b>

- Desirable: <200 mg/dL

- Borderline high: 200-239 mg/dL

- High: ≥240 mg/dL

Your input: {chol} mg/dL

</div>

""", unsafe\_allow\_html=True)

fbs\_options = {

"True": "Fasting glucose >120 mg/dL may indicate diabetes or prediabetes.",

"False": "Normal fasting glucose (<120 mg/dL) reduces diabetes risk."

}

fbs = st.selectbox("Fasting Blood Sugar > 120 mg/dl", list(fbs\_options.keys()))

st.markdown(f"""

<div class="description-box">

<b>ℹ️ Blood Sugar Info:</b> {fbs\_options[fbs]}

</div>

""", unsafe\_allow\_html=True)

with col2:

restecg\_desc = {

"Normal": "No ECG abnormalities detected.",

"ST-T wave abnormality": "May indicate ischemia, electrolyte imbalance, or other conditions.",

"Left ventricular hypertrophy": "Thickened heart muscle, often from high blood pressure."

}

restecg = st.selectbox("Resting ECG Results", list(restecg\_desc.keys()))

st.markdown(f"""

<div class="description-box">

<b>ℹ️ ECG Interpretation:</b> {restecg\_desc[restecg]}

</div>

""", unsafe\_allow\_html=True)

thalach = st.slider('Max Heart Rate Achieved', 71, 202, 150)

max\_hr\_estimate = 220 - age

st.markdown(f"""

<div class="description-box">

<b>ℹ️ Heart Rate Info:</b>

- Your estimated max heart rate: ~{max\_hr\_estimate} bpm (220 - age)

- Your input: {thalach} bpm

- Lower values may indicate reduced cardiovascular fitness

</div>

""", unsafe\_allow\_html=True)

exang\_options = {

"Yes": "Chest pain during exercise suggests possible coronary artery disease.",

"No": "No exercise-induced chest pain is a positive sign."

}

exang = st.selectbox("Exercise Induced Angina", list(exang\_options.keys()))

st.markdown(f"""

<div class="description-box">

<b>ℹ️ Exercise Angina:</b> {exang\_options[exang]}

</div>

""", unsafe\_allow\_html=True)

oldpeak = st.slider('ST Depression', 0.0, 6.2, 2.0, step=0.1)

st.markdown(f"""

<div class="description-box">

<b>ℹ️ ST Depression:</b>

- Measures ECG changes during stress test

- Normal: 0-1 mm

- Mild: 1-2 mm

- Significant: >2 mm

Your input: {oldpeak} mm

</div>

""", unsafe\_allow\_html=True)

slope\_desc = {

"Upsloping": "Generally normal, but context matters.",

"Flat": "May suggest reduced blood flow during stress.",

"Downsloping": "More concerning for significant coronary disease."

}

slope = st.selectbox("Slope of ST Segment", list(slope\_desc.keys()))

st.markdown(f"""

<div class="description-box">

<b>ℹ️ ST Slope:</b> {slope\_desc[slope]}

</div>

""", unsafe\_allow\_html=True)

ca = st.slider('Major Vessels (0-4) Coloured By Flourosopy', 0, 4, 0)

st.markdown(f"""

<div class="description-box">

<b>ℹ️ Vessel Blockage:</b>

Number of major coronary arteries with no significant narrowing seen on angiogram.

Your input: {ca} vessels affected

</div>

""", unsafe\_allow\_html=True)

thal\_desc = {

"Normal": "No blood disorder affecting oxygen transport.",

"Fixed Defect": "Permanent heart muscle damage from prior heart attack.",

"Reversible Defect": "Temporary blood flow issues during stress test."

}

thal = st.selectbox("Thalassemia", list(thal\_desc.keys()))

st.markdown(f"""

<div class="description-box">

<b>ℹ️ Thalassemia Info:</b> {thal\_desc[thal]}

</div>

""", unsafe\_allow\_html=True)

if st.button('🔍 Assess My Risk', type="primary"):

with st.spinner('Analyzing your risk factors...'):

input\_data = preprocess\_input(age, gender, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal)

prediction = predict\_heart\_disease(input\_data)

# Display results

st.subheader("📊 Risk Assessment Results")

if prediction == 1:

st.markdown('<h3 class="risk-high">🚨 High Risk of Heart Disease</h3>', unsafe\_allow\_html=True)

else:

st.markdown('<h3 class="risk-low">✅ Low Risk of Heart Disease</h3>', unsafe\_allow\_html=True)

# Show explanation

st.subheader("🔍 Risk Factors Analysis")

st.markdown(explain\_prediction(prediction, input\_data))

# Show recommendations

st.subheader("💡 Personalized Recommendations")

recommendations = generate\_lifestyle\_recommendations(input\_data, prediction)

for rec in recommendations:

st.markdown(rec)

# PDF Download Section

st.subheader("📄 Download Your Report")

pdf\_data = generate\_pdf(input\_data, prediction, recommendations)

st.download\_button(

label="📥 Download Report as PDF",

data=pdf\_data,

file\_name="heart\_disease\_risk\_report.pdf",

mime="application/pdf"

)

# AI Chatbot section

st.markdown("---")

st.subheader("💬 Heart Health Assistant")

st.write("Ask any questions about heart disease prevention, symptoms, or management:")

user\_query = st.text\_input("Your question:", placeholder="e.g., What are early signs of heart disease?")

if st.button("Ask the AI Assistant") and user\_query:

with st.spinner('Getting expert information...'):

try:

client = openai.OpenAI()

response = client.chat.completions.create(

model="gpt-3.5-turbo",

messages=[

{"role": "system", "content": "You are a heart health assistant. Provide clear, medically accurate information in simple terms."},

{"role": "user", "content": user\_query}

]

)

ai\_response = response.choices[0].message.content.strip()

st.markdown(f"""

<div style="background-color: #092a66; padding: 15px; border-radius: 10px; margin-top: 10px;">

<b>AI Response:</b> {ai\_response}

</div>

""", unsafe\_allow\_html=True)

except Exception as e:

st.error(f"Error accessing AI assistant: {str(e)}")

elif option == "AI Chemist":

st.header("🧪 AI Medication Assistant")

st.markdown("""

<div class="description-box">

Hi there, I am here to help users with medication matters, such as drug interactions, administration instructions,.

You can upload images of medicines and ask me questions related to it.

</div>

""", unsafe\_allow\_html=True)

chem\_input = st.text\_area("Describe your medication query:",

placeholder="e.g., 'What is Dolo-650 used for?'")

uploaded\_image = st.file\_uploader("Upload a chemistry-related image (optional)",

type=["jpg", "png", "jpeg"])

if st.button("Analyze with AI", type="primary"):

if chem\_input.strip() or uploaded\_image:

with st.spinner('Analyzing your chemistry question...'):

try:

image = Image.open(uploaded\_image) if uploaded\_image else None

model = genai.GenerativeModel("models/gemini-1.5-pro-latest")

if image:

response = model.generate\_content([chem\_input, image])

else:

response = model.generate\_content(chem\_input)

st.markdown("### AI Chemist Analysis")

if uploaded\_image:

st.image(image, caption="Uploaded Image", use\_container\_width=True)

st.markdown(f"""

<div style="background-color: #092a66; padding: 15px; border-radius: 10px; margin-top: 10px;">

{response.text}

</div>

""", unsafe\_allow\_html=True)

except Exception as e:

st.error(f"An error occurred: {str(e)}")

else:

st.warning("Please enter a question or upload an image to analyze.")

if \_\_name\_\_ == '\_\_main\_\_':

main()