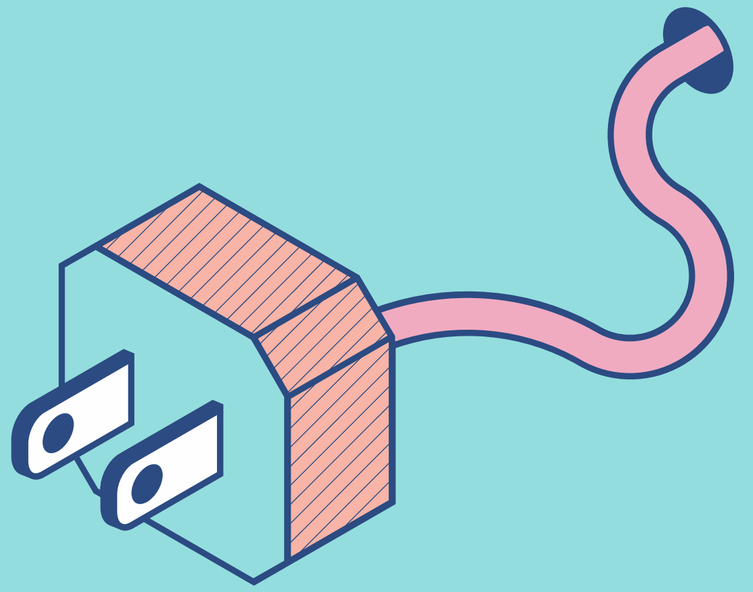
INTELLIGENT HEART RISK PREDICTION

METHODOLOGY

Definition of ANFIS:
The Adaptive Neuro-Fuzzy Inference System (ANFIS) is a hybrid artificial intelligence model that combines fuzzy logic and neural networks. It uses fuzzy logic's reasoning capabilities to handle uncertainty and approximate human decision-making while leveraging neural networks to optimize parameters and improve learning.



Why Choose ANFIS:
ANFIS is chosen for heart disease prediction due to its ability to handle uncertainty in medical data and approximate nonlinear relationships among risk factors. Its hybrid nature allows for high accuracy, interpretability, and adaptability, making it ideal for healthcare applications where precise and reliable predictions are critical.

ASPECT MEASURES

1 AGE

2 SEX

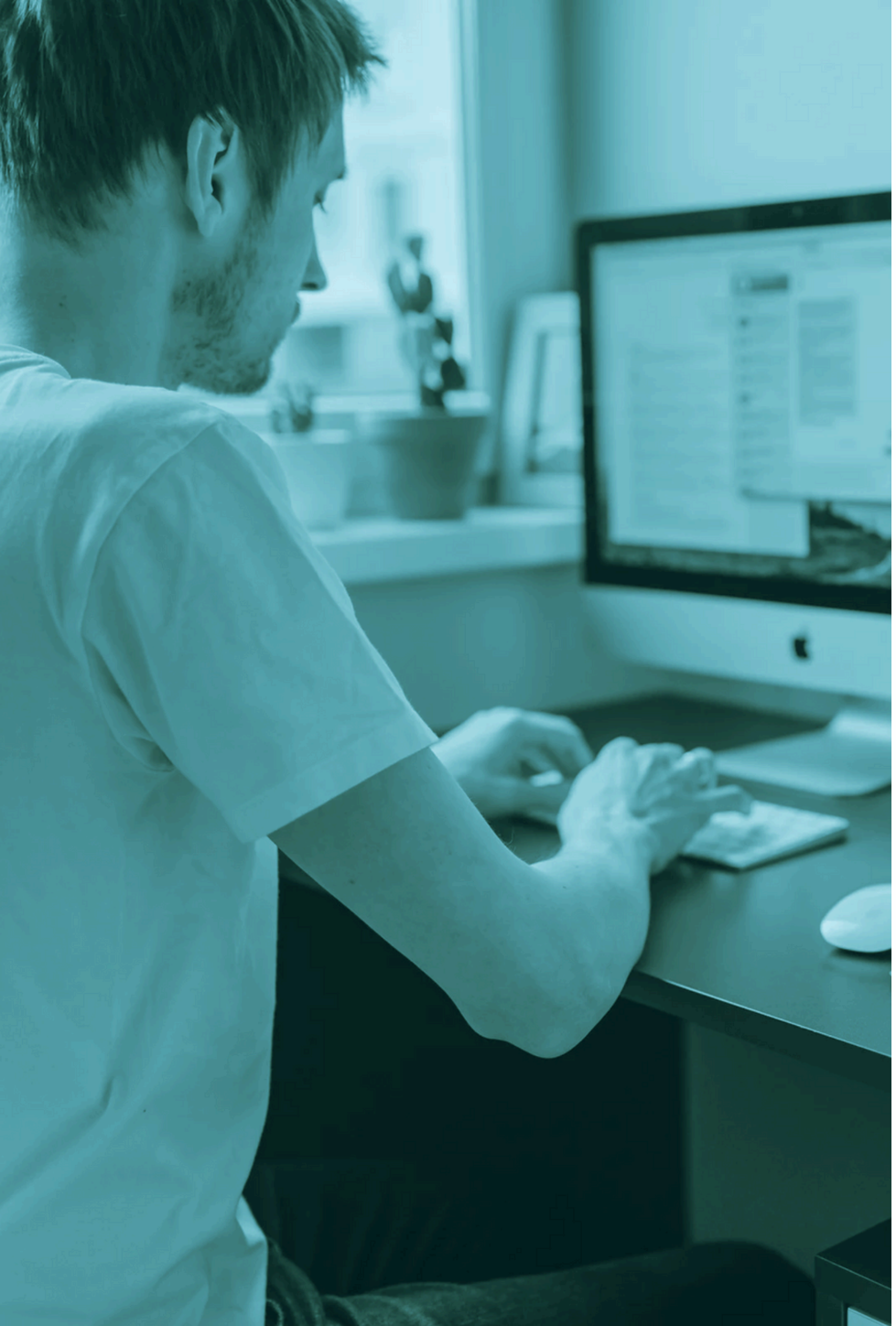
3 CHOLESTROL

4 BLOOD PRESSURE

5 HEART RATE

6 DIABETES

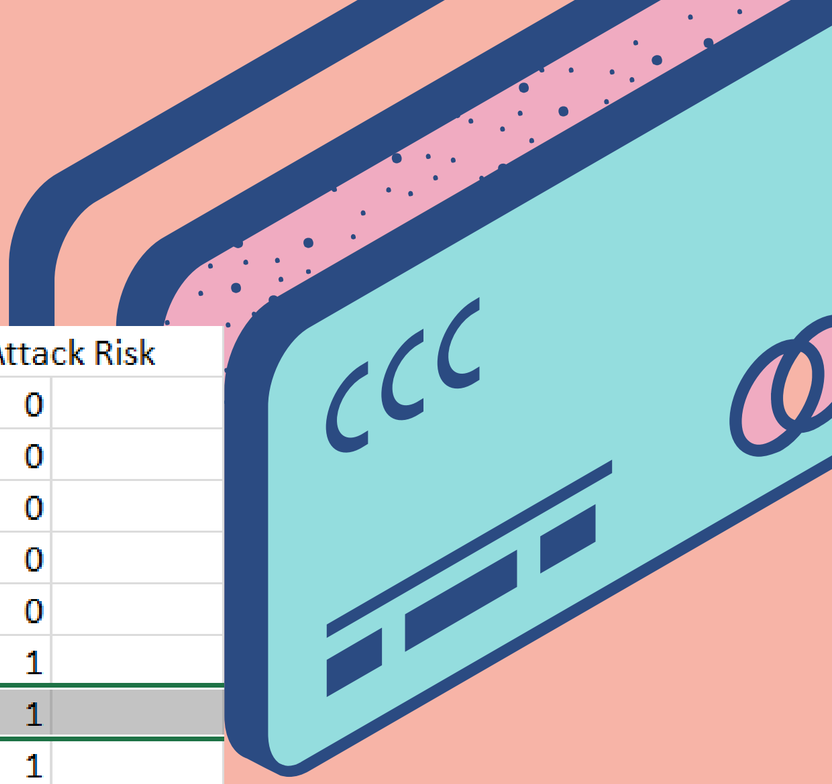
7 DIET



WHAT DO WE DO?

We simulate heart risk prediction using both Python and Matlab. Users are required to initiate input propmts, providing data such as age, sex, cholestrol, blood pressure, heart rate, diabetes and diet. This input will then utilized by neural network (ANFIS) to predict heart risk based on inputs as well as its learned patterns and training.

DATA



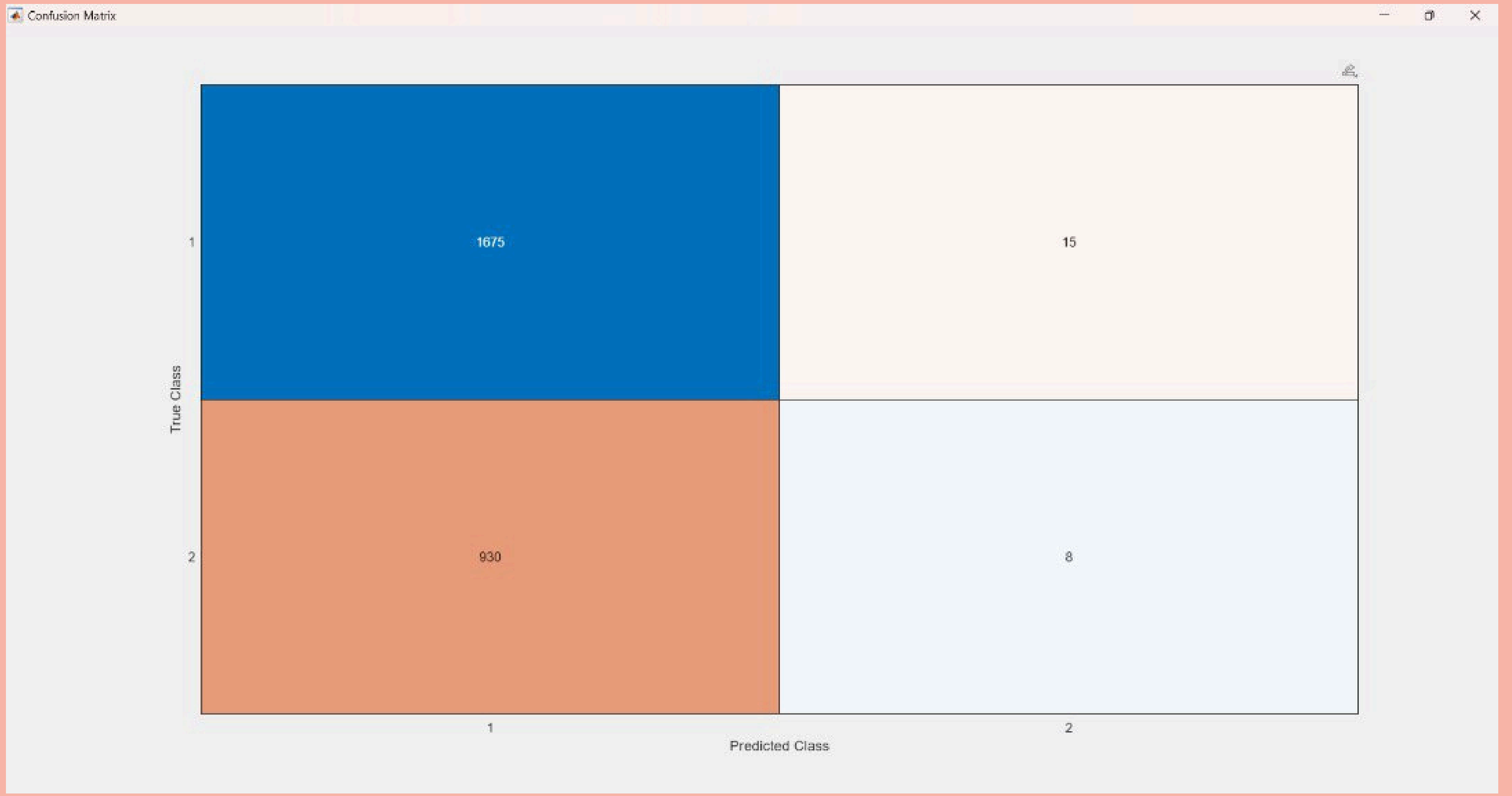
Patient ID	Age	Sex	Cholesterc	Blood Pres	Heart Rate	Diabetes	Diet	Heart Attack Risk
BMW7812	67	Male	208	158/88	72	0	Average	0
CZE1114	21	Male	389	165/93	98	1	Unhealthy	0
BNI9906	21	Female	324	174/99	72	1	Healthy	0
JLN3497	84	Male	383	163/100	73	1	Average	0
GFO8847	66	Male	318	91/88	93	1	Unhealthy	0
ZOO7941	54	Female	297	172/86	48	1	Unhealthy	1
WYV0966	90	Male	358	102/73	84	0	Healthy	1
XXM0972	84	Male	220	131/68	107	0	Average	1
XCQ5937	20	Male	145	144/105	68	1	Average	0
FTJ5456	43	Female	248	160/70	55	0	Unhealthy	0
HSD6283	73	Female	373	107/69	97	1	Average	0
YSP0073	71	Male	374	158/71	70	1	Average	0
FPS0415	77	Male	228	101/72	68	1	Unhealthy	1
YYU9565	60	Male	259	169/72	85	1	Healthy	1
VTW9069	88	Male	297	112/81	102	1	Unhealthy	0
DCY3282	73	Male	122	114/88	97	1	Average	1
DXB2434	69	Male	379	173/75	40	1	Average	0
COP0566	38	Male	166	120/74	56	1	Healthy	0

```
function createHeartRiskPredictionGUI()
    % Create MATLAB GUI for ANFIS Heart Disease Prediction

    % Load data
    dataFile = 'heart_attack_prediction_dataset_Modified.xlsx';
    opts = detectImportOptions(dataFile);
    opts.VariableNamingRule = 'preserve';
    data = readtable(dataFile, opts);

    % Preprocess data (map values, one-hot encoding, etc.)
    [input, output] = preprocessData(data);

    % GUI components
    fig = uifigure('Name', 'Heart Disease Prediction', 'Position', [100, 100, 700

    % Train Button
    trainButton = uibutton(fig, 'Position', [50, 450, 100, 30], 'Text', 'Train AN
                           'ButtonPushedFcn', @(btn, event) trainANFISCallback(in

    % Predict Button
    predictButton = uibutton(fig, 'Position', [200, 450, 100, 30], 'Text', 'Predi
                             'ButtonPushedFcn', @(btn, event) predictCallback(in

    % Axes for displaying results
    ax = uiaxes(fig, 'Position', [50, 50, 300, 300]);
    title(ax, 'Prediction Results');
```

```
    % Confusion Matrix Display Area
    confusionMatrixPanel = uipanel(fig, 'Position', [400, 50, 250, 250], 'Title',
    confusionAxes = uiaxes(confusionMatrixPanel, 'Position', [20, 20, 200, 200]);

    % Status Label
    statusLabel = uilabel(fig, 'Position', [400, 450, 200, 30], 'Text', 'Status:

    % Callback functions
    function trainANFISCallback(input, output)
        try
            % Split data into train and test
            [XTrain, YTrain, XTest, YTest] = splitData(input, output);

            % Train ANFIS
            fis = anfis([XTrain YTrain], genfis1([XTrain YTrain]));

            % Save model
            assignin('base', 'fisModel', fis);
            statusLabel.Text = 'Status: Model Trained';

            % Evaluate on test data
            predictions = evalfis(fis, XTest);

            % Binarize predictions (e.g., threshold at 0.5)
            predictionsBinary = predictions >= 0.5;
            YTestBinary = YTest >= 0.5;
```

```
            % Simulate user input for prediction
            % (You can implement additional fields for user input as before)
            prediction = evalfis(fis, input);  % Example using entire dataset
            statusLabel.Text = 'Status: Prediction Completed';

            % Display results in the graph
            scatter(ax, output, prediction);
            xlabel(ax, 'True Values'); ylabel(ax, 'Predicted Values');
        catch
            statusLabel.Text = 'Status: No Model Found or Invalid Input';
        end
    end
end

function [input, output] = preprocessData(data)
    % Implement data preprocessing steps
    Age = data{:, 1};
    Gender = dummyvar(categorical(data{:, 2}));
    Cholesterol = data{:, 3};
    BloodPressure = calculateMAP(data{:, 4});

    % Other preprocessing steps...
    input = [Age Gender Cholesterol BloodPressure]; % Example features
    output = data{:, 25}; % Target variable
end
```

```
function mapValues = calculateMAP(bpValues)
    % Process blood pressure data
    mapValues = zeros(length(bpValues), 1);
    for i = 1:length(bpValues)
        bpSplit = strsplit(bpValues{i}, '/');
        systolic = str2double(bpSplit{1});
        diastolic = str2double(bpSplit{2});
        mapValues(i) = (2 * diastolic + systolic) / 3;
    end
end

function [XTrain, YTrain, XTest, YTest] = splitData(input, output)
    % Split data into train and test sets
    cv = cvpartition(size(input, 1), 'HoldOut', 0.3);
    idx = cv.test;
    XTrain = input(~idx, :);
    YTrain = output(~idx);
    XTest = input(idx, :);
    YTest = output(idx);
end
```


Confusion Matrix
1675
15
930
8
1
2
True Class
1
2
Predicted Class

CODING

```python
import tkinter as tk
from tkinter import ttk, messagebox
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report


# Define Gaussian Membership Function
def gaussmf(x, mean, sigma):
    return np.exp(-((x - mean) ** 2) / (2 * sigma ** 2))


# Function to simulate dataset if file is not available
def create_sample_dataset(file_path):
    data = {
        "Age": np.random.randint( low: 20,  high: 80,  size: 1000),
        "Sex": np.random.choice( a: ["Male", "Female"],  size: 1000),
        "Cholesterol": np.random.randint( low: 150,  high: 300,  size: 1000),
        "Blood Pressure": [f"{np.random.randint( low: 100,  high: 140)}/{np.random.randint( low: 60,  high: 90)}" for _ in range(1000)],
        "Heart Rate": np.random.randint( low: 60,  high: 120,  size: 1000),
        "Diabetes": np.random.choice( a: ["Yes", "No"],  size: 1000),
        "Diet": np.random.choice( a: ["Healthy", "Average", "Unhealthy"],  size: 1000),
        "Heart Attack Risk": np.random.choice( a: [0, 1],  size: 1000)
    }
    df = pd.DataFrame(data)
    df.to_excel(file_path, index=False, sheet_name="heart_attack_prediction_dataset")

```

```python
# File path to the dataset
file_path = "heart_attack_prediction_dataset.xlsx"

# Check if file exists; if not, create a sample dataset
try:
    data = pd.ExcelFile(file_path).parse('heart_attack_prediction_dataset')
except FileNotFoundError:
    create_sample_dataset(file_path)
    data = pd.ExcelFile(file_path).parse('heart_attack_prediction_dataset')

# Preprocess data
columns = ['Age', 'Sex', 'Cholesterol', 'Blood Pressure', 'Heart Rate', 'Diabetes', 'Diet', 'Heart Attack Risk']
data_filtered = data[columns].copy()
data_filtered['Sex'] = data_filtered['Sex'].map({"Male": 0, "Female": 1})
data_filtered['Diabetes'] = data_filtered['Diabetes'].map({"Yes": 1, "No": 0})
data_filtered['Diet'] = data_filtered['Diet'].map({"Healthy": 0, "Average": 1, "Unhealthy": 2})


# Convert Blood Pressure from 'systolic/diastolic' to systolic values
def extract_systolic(bp):
    try:
        systolic = int(bp.split('/')[0])
        return systolic
    except:
        return np.nan


data_filtered['Blood Pressure'] = data_filtered['Blood Pressure'].apply(extract_systolic)
data_filtered.dropna(subset=['Blood Pressure'], inplace=True)
```

```python
# Define features and target
X = data_filtered[['Age', 'Sex', 'Cholesterol', 'Blood Pressure', 'Heart Rate', 'Diabetes', 'Diet']]
y = data_filtered['Heart Attack Risk']

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


# Define membership functions
def define_membership_functions():
    return {
        'Age': [
            ('young', gaussmf, {'mean': 25, 'sigma': 10}),
            ('middle-aged', gaussmf, {'mean': 45, 'sigma': 10}),
            ('old', gaussmf, {'mean': 65, 'sigma': 10}),
        ],
        'Cholesterol': [
            ('low', gaussmf, {'mean': 150, 'sigma': 30}),
            ('normal', gaussmf, {'mean': 200, 'sigma': 30}),
            ('high', gaussmf, {'mean': 250, 'sigma': 30}),
        ],
        'Blood Pressure': [
            ('low', gaussmf, {'mean': 110, 'sigma': 10}),
            ('normal', gaussmf, {'mean': 120, 'sigma': 10}),
            ('high', gaussmf, {'mean': 140, 'sigma': 10}),
        ],
        'Heart Rate': [
            ('low', gaussmf, {'mean': 60, 'sigma': 10}),
            ('normal', gaussmf, {'mean': 80, 'sigma': 10}),
            ('high', gaussmf, {'mean': 100, 'sigma': 10}),
        ],
        'Diet': [
            ('healthy', gaussmf, {'mean': 0, 'sigma': 1}),
            ('average', gaussmf, {'mean': 1, 'sigma': 1}),
            ('unhealthy', gaussmf, {'mean': 2, 'sigma': 1}),
        ],
    }


# Placeholder for ANFIS model (to be implemented with custom logic or an alternative library)
class ANFISModel:
    def __init__(self, membership_functions):
        self.membership_functions = membership_functions

    def fit(self, X, y, epochs=50, learning_rate=0.01):
        print("Training ANFIS model (placeholder)...")
        # Add your custom training logic here

    def predict(self, X):
        print("Predicting with ANFIS model (placeholder)...")
        return np.random.choice( a: [0, 1], size=X.shape[0])  # Placeholder logic


membership_functions = define_membership_functions()
model = ANFISModel(membership_functions)

# Fit the ANFIS model
try:
    model.fit(X_scaled, y, epochs=50, learning_rate=0.01)
except Exception as e:
    print(f"Error during training: {e}")
```

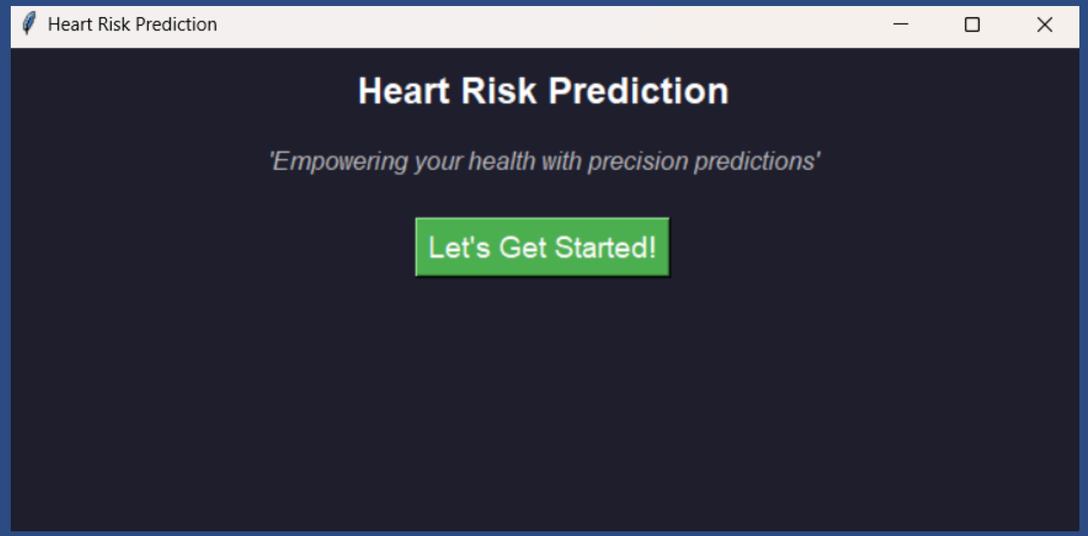
```
        self.entries = {}

        for idx, field in enumerate(fields):
            tk.Label(start_window, text=field, bg="#1e1e2f", fg="#ffffff").pack(anchor='w', padx=20)
            entry = tk.Entry(start_window)
            entry.pack(pady=5, padx=20, anchor='w')
            self.entries[field] = entry

        # Predict button
        tk.Button(start_window, text="Predict", command=self.predict).pack(pady=20)

    def predict(self):
        try:
            # Collect inputs
            age = int(self.entries["Age"].get())
            gender_input = self.entries["Sex (Male/Female)"].get().strip().lower()
            if gender_input == 'male':
                gender = 0
            elif gender_input == 'female':
                gender = 1
            else:
                raise ValueError("Invalid gender. Please enter Male or Female.")

            cholesterol = float(self.entries["Cholesterol"].get())

            # Process Blood Pressure
            blood_pressure_input = self.entries["Blood Pressure"].get().strip()
            blood_pressure = extract_systolic(blood_pressure_input)
            if np.isnan(blood_pressure):
                raise ValueError("Invalid blood pressure format. Use systolic/diastolic (e.g., 120/80).")
```

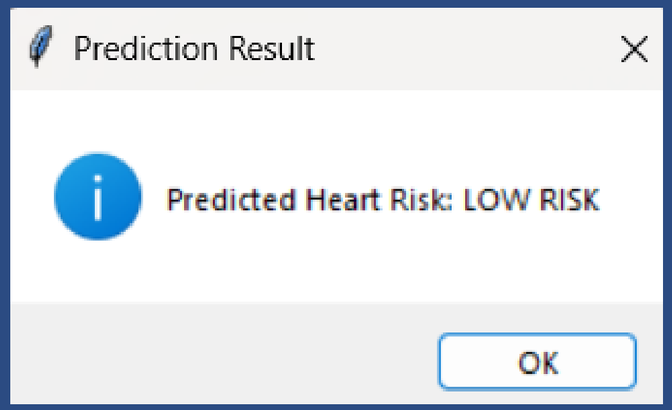
```
# GUI Implementation
class HeartAttackPredictionGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Heart Risk Prediction")
        self.root.geometry("700x600")
        self.root.configure(bg="#1e1e2f")

        # Title
        tk.Label(self.root, text="Heart Risk Prediction", font=('Futura', 18, 'bold'), bg="#1e1e2f", fg="#ffffff").pack(
            pady=10)

        self.label = tk.Label(self.root, text="'Empowering your health with precision predictions'",
                              font=('Futura', 12, 'italic'), bg="#1e1e2f", fg="#c0c0c0")  # Light gray text
        self.label.pack(padx=20, pady=5)

        tk.Button(self.root, text="Let's Get Started!", font=('Futura', 14), bg="#4caf50", fg="#ffffff",
                  command=self.start).pack(pady=20)
        self.root.mainloop()

    def start(self):
        start_window = tk.Toplevel(self.root)
        start_window.title("Prediction")
        start_window.geometry("700x600")
        start_window.configure(bg="#1e1e2f")

        tk.Label(start_window, text="Fill out the following details:", font=('Futura', 16), bg="#1e1e2f",
                 fg="#ffffff").pack(pady=10)

        # Define form fields
        fields = ["Name", "Age", "Sex (Male/Female)", "Cholesterol", "Blood Pressure", "Heart Rate",
                  "Diabetes (Yes/No)", "Diet"]
```

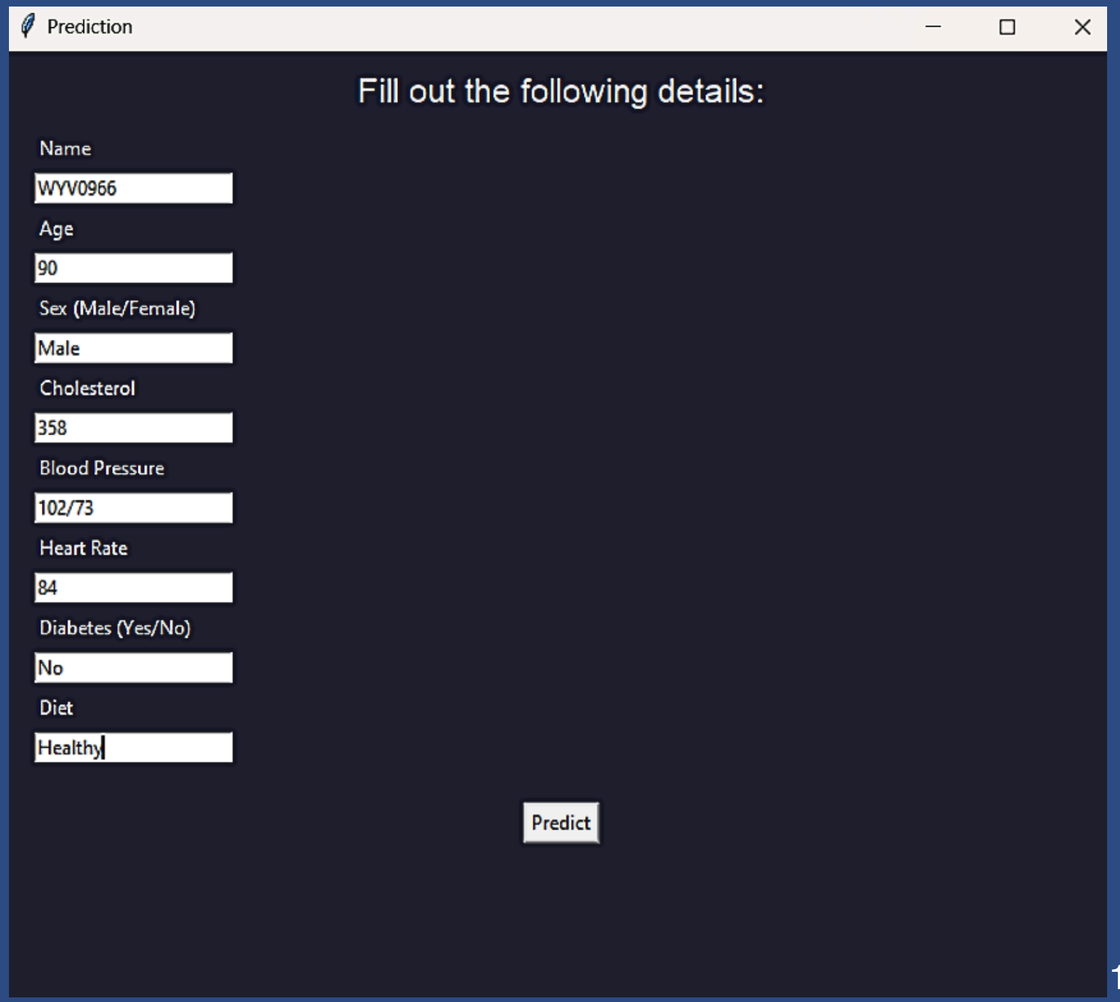
```
        self.entries = {}

        for idx, field in enumerate(fields):
            tk.Label(start_window, text=field, bg="#1e1e2f", fg="#ffffff").pack(anchor='w', padx=20)
            entry = tk.Entry(start_window)
            entry.pack(pady=5, padx=20, anchor='w')
            self.entries[field] = entry

        # Predict button
        tk.Button(start_window, text="Predict", command=self.predict).pack(pady=20)

    def predict(self):
        try:
            # Collect inputs
            age = int(self.entries["Age"].get())
            gender_input = self.entries["Sex (Male/Female)"].get().strip().lower()
            if gender_input == 'male':
                gender = 0
            elif gender_input == 'female':
                gender = 1
            else:
                raise ValueError("Invalid gender. Please enter Male or Female.")

            cholesterol = float(self.entries["Cholesterol"].get())

            # Process Blood Pressure
            blood_pressure_input = self.entries["Blood Pressure"].get().strip()
            blood_pressure = extract_systolic(blood_pressure_input)
            if np.isnan(blood_pressure):
                raise ValueError("Invalid blood pressure format. Use systolic/diastolic (e.g., 120/80).")
```

```
            heart_rate = float(self.entries["Heart Rate"].get())
            diabetes = 1 if self.entries["Diabetes (Yes/No)"].get().strip().lower() == 'yes' else 0

            # Process Diet
            diet_input = self.entries["Diet"].get().strip().capitalize()
            diet_mapping = {"Healthy": 0, "Average": 1, "Unhealthy": 2}
            if diet_input in diet_mapping:
                diet = diet_mapping[diet_input]
            else:
                raise ValueError("Invalid diet. Please select Healthy, Average, or Unhealthy.")

            # Scale input
            input_data = scaler.transform([[age, gender, cholesterol, blood_pressure, heart_rate, diabetes, diet]])

            # Predict using ANFIS (placeholder prediction)
            prediction = model.predict(input_data)[0]
            result = "HIGH RISK" if prediction == 1 else "LOW RISK"

            # Display result
            messagebox.showinfo( title: "Prediction Result",  message: f"Predicted Heart Risk: {result}")

        except Exception as e:
            messagebox.showerror( title: "Error",  message: f"An error occurred: {e}")


HeartAttackPredictionGUI()
```

GRAPHICAL USER INTERFACE







Thank you!