Heart Disease Detection Project

1. Project Description

This project aims to develop a **Heart Disease Detection System** using **both a rule-based expert system (Experta) and a machine learning model (Decision Tree Classifier in Scikit-Learn)**. The system will analyze patient health indicators to predict heart disease risk. Additionally, it includes **data preprocessing, visualization, and an organized folder structure** to ensure clarity and usability.

2. Requirements & Implementation Steps

Step 1: Dataset Processing

- **Load Dataset**: Use Pandas to read the heart disease dataset.
- **Handle Missing Values**: Fill missing values using mean/median or drop incomplete rows.
- **Normalize Data**: Scale numerical features (e.g., blood pressure, cholesterol) using MinMaxScaler.
- **Encode Categorical Variables**: Convert categorical features into numerical values using One-Hot Encoding.
- **Feature Selection**: Identify the most important features for prediction using correlation analysis.
- **Save Cleaned Data**: Store the cleaned dataset as `cleaned_data.csv`.

Step 2: Data Visualization

- **Statistical Summary**: Display distributions of key features using Pandas and Seaborn.
- **Correlation Heatmap**: Use Seaborn to visualize feature correlations.
- **Histograms & Boxplots**: Analyze data distribution and outliers.
- **Feature Importance Plot**: Rank features based on significance for heart disease prediction.

Step 3: Implement Rule-Based Expert System (Experta)

- **Define At Least 10 Rules**:
  - Example: If `Cholesterol > 240` and `Age > 50`, risk = high.
  - Example: If `BloodPressure > 140` and `Smoking = Yes`, risk = high.
  - Example: If `Exercise = Regular` and `BMI < 25`, risk = low.
- **Create Knowledge Base**: Store the rules in an Experta-based inference engine.

- **Inference Mechanism**: Implement rule-firing mechanism for risk assessment.
- **User Input Support**: Allow users to input symptoms and get a risk prediction.

Step 4: Build Decision Tree Model (Scikit-Learn)

- **Split Data**: Use an 80/20 train-test split.
- **Train Model**: Train a Decision Tree Classifier on the dataset.
- **Hyperparameter Tuning**: Optimize tree depth and min samples per split.
- **Evaluate Model**: Measure performance using accuracy, precision, recall, and F1-score.
- **Save Model**: Export the trained model using `joblib`.

Step 5: Compare Expert System and Decision Tree Model

- **Validation Set Evaluation**: Test both systems on unseen data.
- **Accuracy Comparison**: Compare performance metrics.
- **Explainability**: Analyze decision trees vs. human-defined rules.

Step 6: Integration & GitHub Upload

- **Organize Codebase**: Structure files into logical folders.
- **Write Documentation**: Provide setup instructions and usage examples.
- **Push to GitHub**: Ensure a clean repository with a README file.

3. Deliverables

- **Cleaned and Preprocessed Heart Disease Dataset**
- **Data Visualization Notebook** (with insights and analysis)
- **Rule-Based Expert System** using Experta (with at least 10 rules)
- **Decision Tree Model** using Scikit-Learn (with hyperparameter tuning)
- **Accuracy Comparison Report**
- **Structured Codebase** for easy navigation
- **Project Documentation** (README with instructions)

4. Folder Structure

```
Heart_Disease_Detection/

|── data/              # Contains the dataset (raw & cleaned)

│   ├── raw_data.csv

│   ├── cleaned_data.csv

|── notebooks/         # Jupyter Notebooks for visualization & preprocessing
```

```
│   ├── data_analysis.ipynb
│   ├── model_training.ipynb
│── rule_based_system/      # Rule-based system using Experta
│   ├── rules.py
│   ├── expert_system.py
│── ml_model/               # Decision Tree implementation
│   ├── train_model.py
│   ├── predict.py
│── utils/                  # Helper functions for data cleaning & processing
│   ├── data_processing.py
│── reports/                # Comparison reports and evaluation
│   ├── accuracy_comparison.md
│── ui/                     # Streamlit UI for user interaction
│   ├── app.py
│── README.md               # Project documentation & setup instructions
│── requirements.txt        # List of dependencies
```

5. Bonus Features

Interactive UI with Streamlit

- **User-Friendly Interface**: Design a web-based UI for risk prediction.
- **Model Integration**: Allow users to input health data and receive a risk assessment.
- **Visualization Dashboard**: Show charts and stats dynamically.
- **Deployable App**: Package the Streamlit app for easy access.