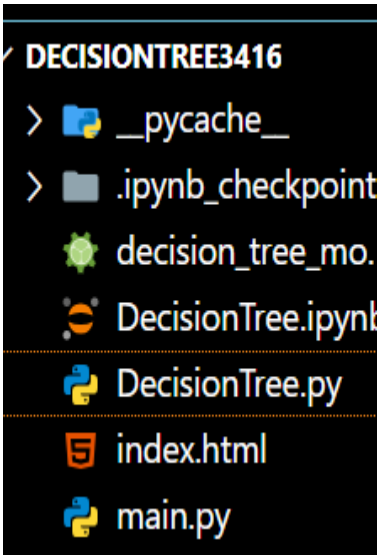
Decision Tree Model Integrated with Website using FastAPI

Name - Barsha Baibhabi
Roll No - 22053416

1. Overview

This project implements a Decision Tree model for predicting diabetes progression using the Diabetes dataset from `sklearn.datasets`. The dataset consists of 10 numerical features that help predict a continuous target value representing disease progression. The trained model is deployed using FastAPI, and a simple HTML frontend allows users to input feature values and get predictions in real-time.

Files in the Project:



1. **`DecisionTree.ipynb`** – Trains the model and saves it as a pickle file.
2. **`main.py`** – FastAPI backend for handling predictions.
3. **`index.html`** – Web frontend for user input and displaying results.
4. **Decision_tree_model.pkl** - Saved logistic regression models.
5. **DecisionTree.py -** Python script version of the Jupyter Notebook.

2. Installation & Setup

Prerequisites

Ensure you have Python 3 installed. Install the required dependencies:

```
pip install fastapi uvicorn scikit-learn numpy pandas pickle-mixin
```

Running the Application

1. Train the model and generate a pickle file by running `decision_tree_model.ipynb`.
2. Start the FastAPI backend:

   ```
   uvicorn main:app --host 127.0.0.1 --port 8006 --reload
   ```

3. Open `index.html` in a browser and test predictions.

3. Training the Model & Generating Pickle File

The `decision_tree_model.ipynb` notebook does the following:

- Loads the Diabetes dataset from `sklearn.datasets`.
- Splits the data into training and test sets.
- Trains a Decision Tree Regressor.
- Saves the trained model as a pickle file (`decision_tree_model.pkl`).

4. FastAPI Backend (`main.py`)

The backend is implemented using FastAPI to:

- Load the trained model from `decision_tree_model.pkl`.

```
@app.post("/predict")

def predict(request: PredictionRequest):

    try:

        features = np.array(request.features).reshape(1, -1)

        prediction = model.predict(features)

        return {"prediction": prediction.tolist()}

    except Exception as e:

        raise HTTPException(status_code=400, detail=str(e))
```

```
if __name__ == "__main__":

    import uvicorn

    uvicorn.run(app, host="0.0.0.0", port=8006)
```

- Accept feature inputs from the frontend via a POST request.
- Process input features and return the predicted diabetes progression value.
- Runs on `http://127.0.0.1:8006/`

5. Frontend (`index.html`) with Inline CSS & JavaScript

The frontend:

- Provides an input form with 10 fields for feature values.
- Uses JavaScript to send AJAX requests to the FastAPI backend.

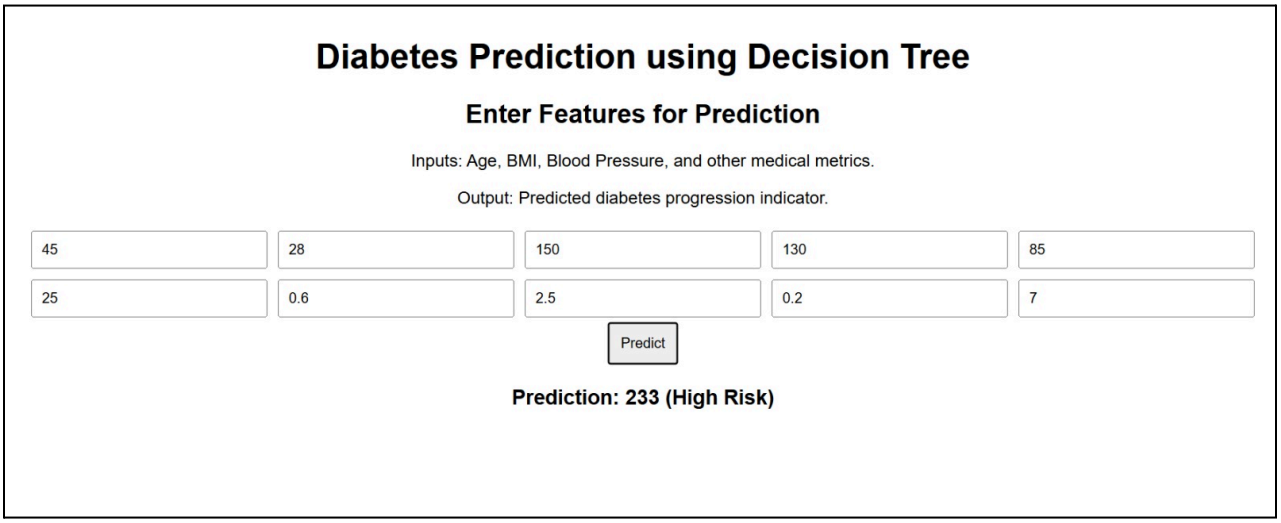
```
function predict() {

            const inputs = document.querySelectorAll("input");

            const features = Array.from(inputs).map(input =>
parseFloat(input.value) || 0);



            fetch("http://localhost:8006/predict", {

                method: "POST",

                headers: { "Content-Type": "application/json" },

                body: JSON.stringify({ features })

            })

            .then(response => response.json())

            .then(data => {

                let prediction = data.prediction;
```

```
                let riskLevel = prediction > 150 ? "High Risk" : "Low
Risk";

                document.getElementById("result").innerText = "Prediction:
" + prediction + " (" + riskLevel + ")";

            })

            .catch(error => {

                document.getElementById("result").innerText = "Error: " +
error;

            });

        }
```

- Displays the predicted progression value, categorized as "High" or "Low" based on a threshold.

6. Testing the Integration

- Enter 10 numerical feature values in `index.html`.
- Click **Predict** to send a request to `FastAPI`.
- The API returns the predicted diabetes progression score.
- The result is displayed on the webpage as "High" or "Low".



7. Conclusion

This project successfully integrates a Decision Tree regression model with a FastAPI backend and a simple web frontend. It provides a foundation for further enhancements, such as improved UI, database integration, and additional model tuning.