Decision Tree Model for Diabetes Prediction

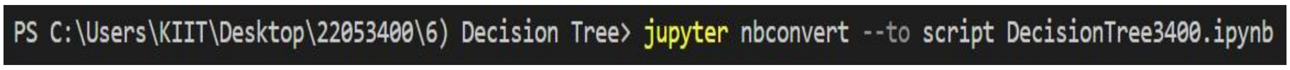
NAME: Ankita Satapathy
ROLL NO: 22053400

1. Introduction

This project implements a custom Decision Tree model to predict diabetes risk based on selected health indicators. The model is built using Python, without relying on predefined implementations from sklearn. The application includes a FastAPI-based backend and an interactive web interface.

2. Files in the Submission

i. DecisionTree3400.ipynb: Implements the custom Decision Tree model in Jupyter Notebook, detailing the training process.
ii. DecisionTree3400.py: Python script implementing the custom Decision Tree classifier.



```
PS C:\Users\KIIT\Desktop\22053400\6) Decision Tree> jupyter nbconvert --to script DecisionTree3400.ipynb
```

iii. main.py: Defines the FastAPI backend to handle prediction requests.
iv. custom_decision_tree.pkl: Serialized trained model using pickle.
v. index.html: Front-end UI for user input and prediction display.

3. Installation and Setup

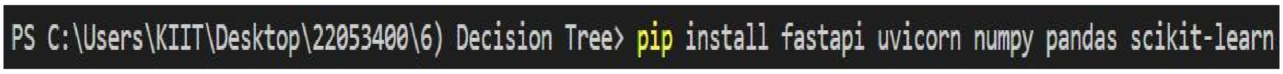
i. Prerequisites

Required Python packages: numpy, scikit-learn, fastapi, pickle, uvicorn.

- NumPy: Used for numerical computations.
- Scikit-learn: Provides dataset utilities for loading diabetes data.
- FastAPI: Used to build a web API for serving predictions.
- Pickle: Used to save and load the trained model.
- Uvicorn: Runs the FastAPI server asynchronously.

ii. Installing Dependencies

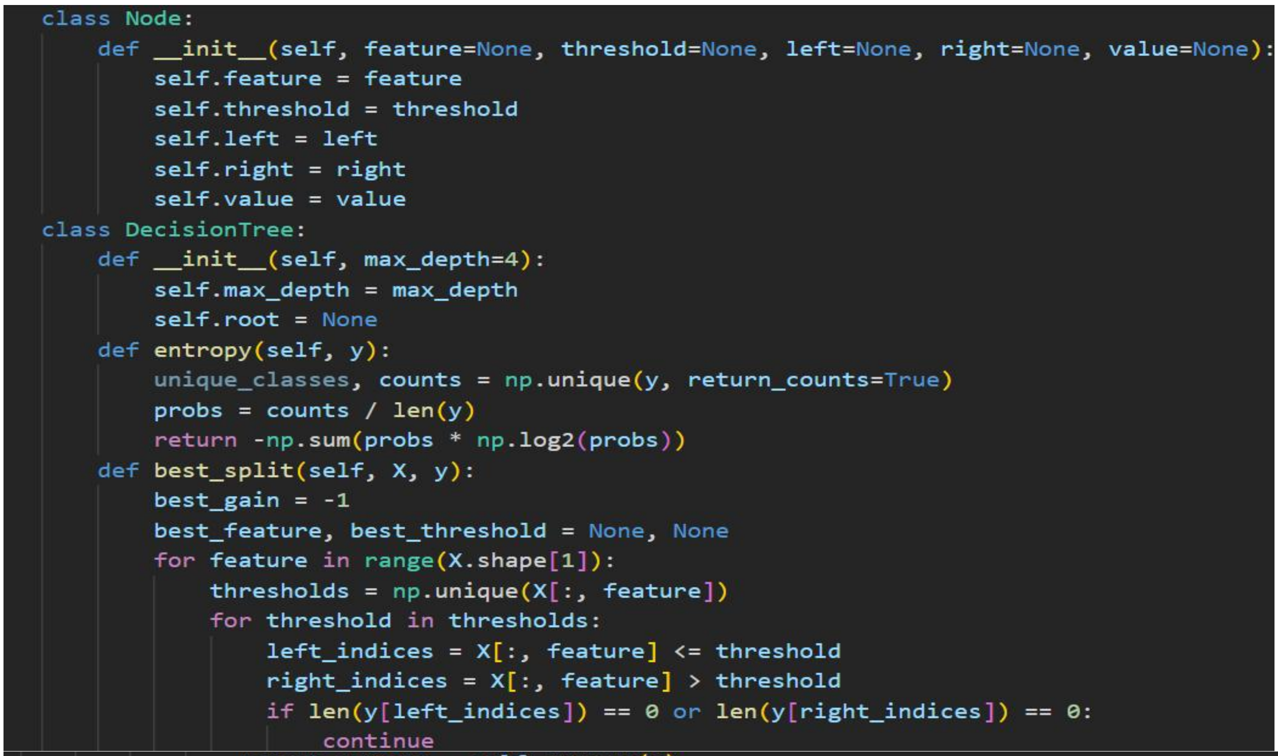
Run the following command in the VSCode terminal to install dependencies:



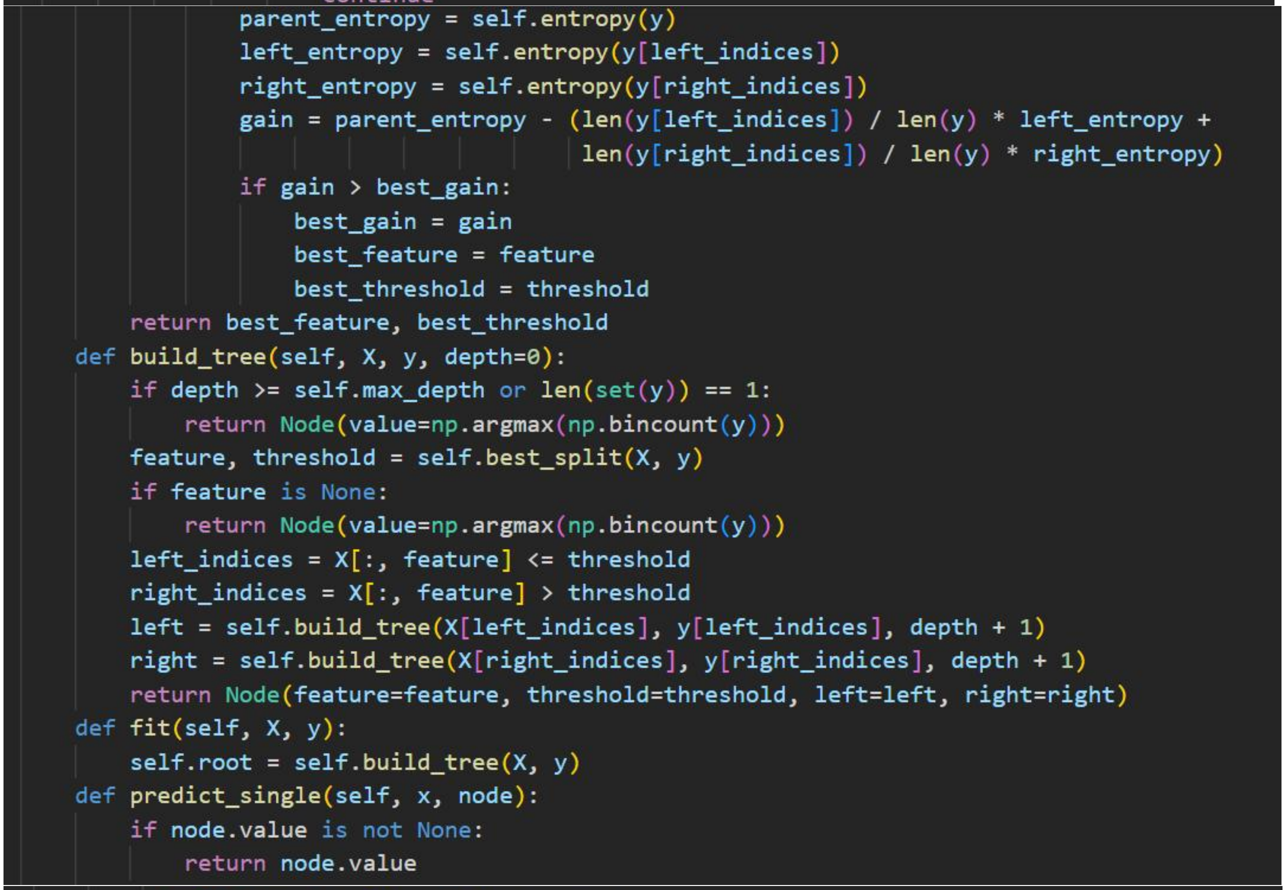
```
PS C:\Users\KIIT\Desktop\22053400\6) Decision Tree> pip install fastapi uvicorn numpy pandas scikit-learn
```

4. Model Implementation

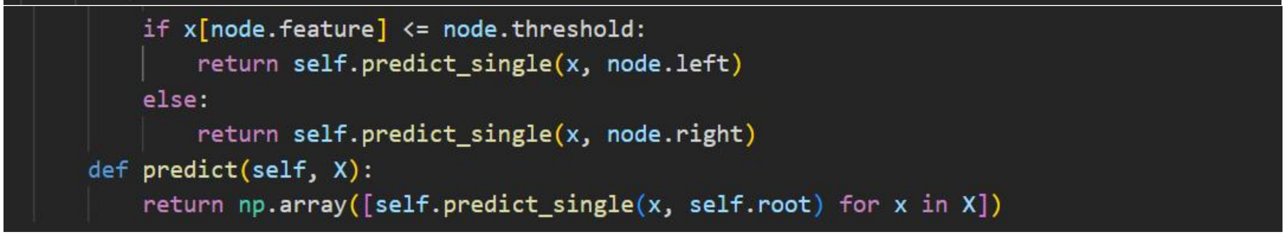
The DecisionTree class in DecisionTree3400.py implements a custom decision tree classifier using entropy-based splitting and recursive tree construction.



```python
class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None, value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
class DecisionTree:
    def __init__(self, max_depth=4):
        self.max_depth = max_depth
        self.root = None
    def entropy(self, y):
        unique_classes, counts = np.unique(y, return_counts=True)
        probs = counts / len(y)
        return -np.sum(probs * np.log2(probs))
    def best_split(self, X, y):
        best_gain = -1
        best_feature, best_threshold = None, None
        for feature in range(X.shape[1]):
            thresholds = np.unique(X[:, feature])
            for threshold in thresholds:
                left_indices = X[:, feature] <= threshold
                right_indices = X[:, feature] > threshold
                if len(y[left_indices]) == 0 or len(y[right_indices]) == 0:
                    continue
```



```python
                parent_entropy = self.entropy(y)
                left_entropy = self.entropy(y[left_indices])
                right_entropy = self.entropy(y[right_indices])
                gain = parent_entropy - (len(y[left_indices]) / len(y) * left_entropy +
                                         len(y[right_indices]) / len(y) * right_entropy)
                if gain > best_gain:
                    best_gain = gain
                    best_feature = feature
                    best_threshold = threshold
        return best_feature, best_threshold
    def build_tree(self, X, y, depth=0):
        if depth >= self.max_depth or len(set(y)) == 1:
            return Node(value=np.argmax(np.bincount(y)))
        feature, threshold = self.best_split(X, y)
        if feature is None:
            return Node(value=np.argmax(np.bincount(y)))
        left_indices = X[:, feature] <= threshold
        right_indices = X[:, feature] > threshold
        left = self.build_tree(X[left_indices], y[left_indices], depth + 1)
        right = self.build_tree(X[right_indices], y[right_indices], depth + 1)
        return Node(feature=feature, threshold=threshold, left=left, right=right)
    def fit(self, X, y):
        self.root = self.build_tree(X, y)
    def predict_single(self, x, node):
        if node.value is not None:
            return node.value
```



```python
        if x[node.feature] <= node.threshold:
            return self.predict_single(x, node.left)
        else:
            return self.predict_single(x, node.right)
    def predict(self, X):
        return np.array([self.predict_single(x, self.root) for x in X])
```

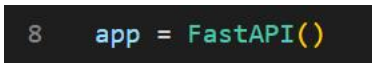
5. Training the Model

- The dataset used is the Diabetes Dataset from sklearn.datasets.
- The model is trained using Age, BMI, Glucose Level, and Blood Pressure as features.
- Target values are converted into binary classes (High Risk/Low Risk).
- The model is trained using a recursive splitting approach based on entropy calculations.
- After training, the model is serialized using pickle for future use

6. API Implementation

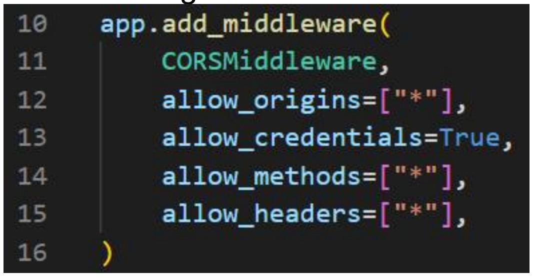
The FastAPI-based backend (main.py) loads the trained model and provides an endpoint to make predictions:

i. API Setup



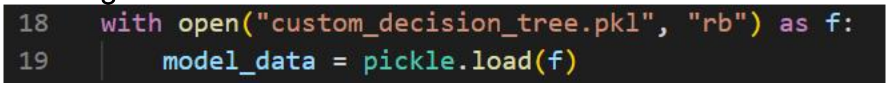
```python
app = FastAPI()
```

ii. CORS Configuration- To allow front-end requests:



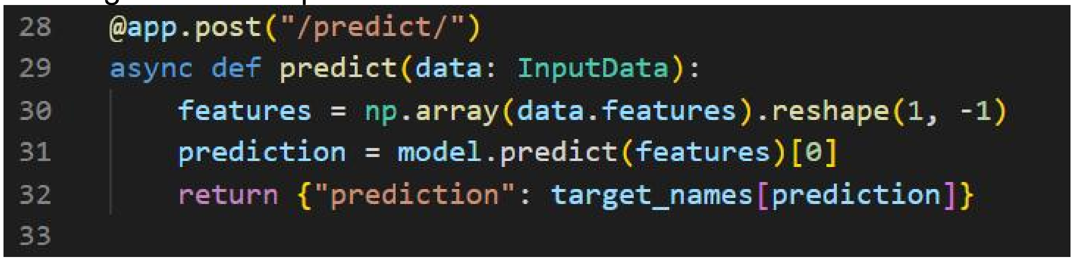
```python
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

iii. Loading the Trained Model



```python
with open("custom_decision_tree.pkl", "rb") as f:
    model_data = pickle.load(f)
```

iv. Defining the API Endpoint

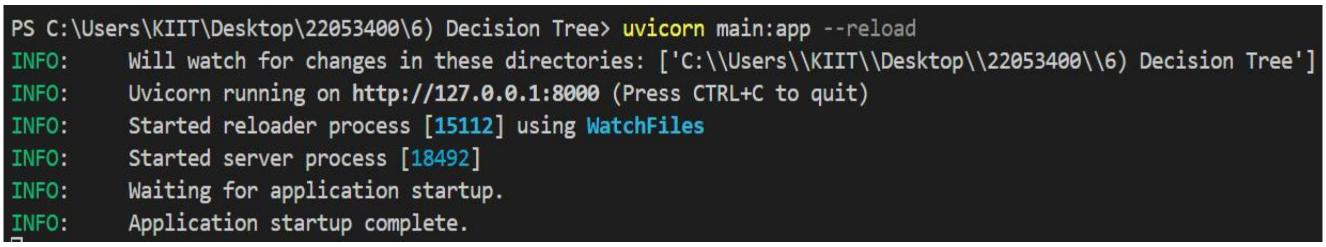


```python
@app.post("/predict/")
async def predict(data: InputData):
    features = np.array(data.features).reshape(1, -1)
    prediction = model.predict(features)[0]
    return {"prediction": target_names[prediction]}

```

7. Running the Application (FastAPI Server)

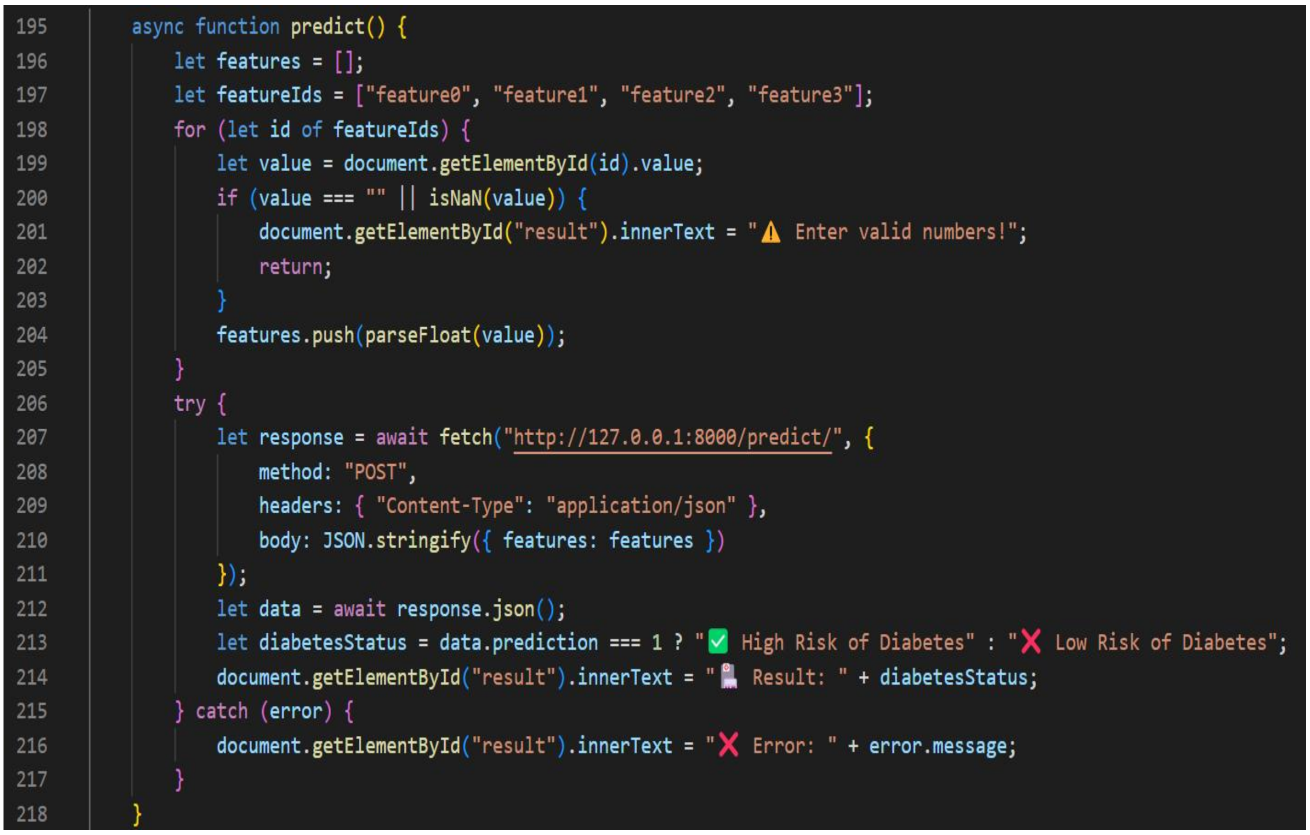
To start the FastAPI server, run the following command in VSCode terminal:



```
PS C:\Users\KIIT\Desktop\22053400\6) Decision Tree> uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\KIIT\\Desktop\\22053400\\6) Decision Tree']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [15112] using WatchFiles
INFO:     Started server process [18492]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

8. Front-End Implementation

The index.html file provides an interactive web interface where users can enter health metrics (Age, BMI, Glucose Level, Blood Pressure) to check their diabetes risk. It includes inline CSS for styling and JavaScript for API communication. When the user submits values, an request is sent to the FastAPI backend, and the result is displayed dynamically.

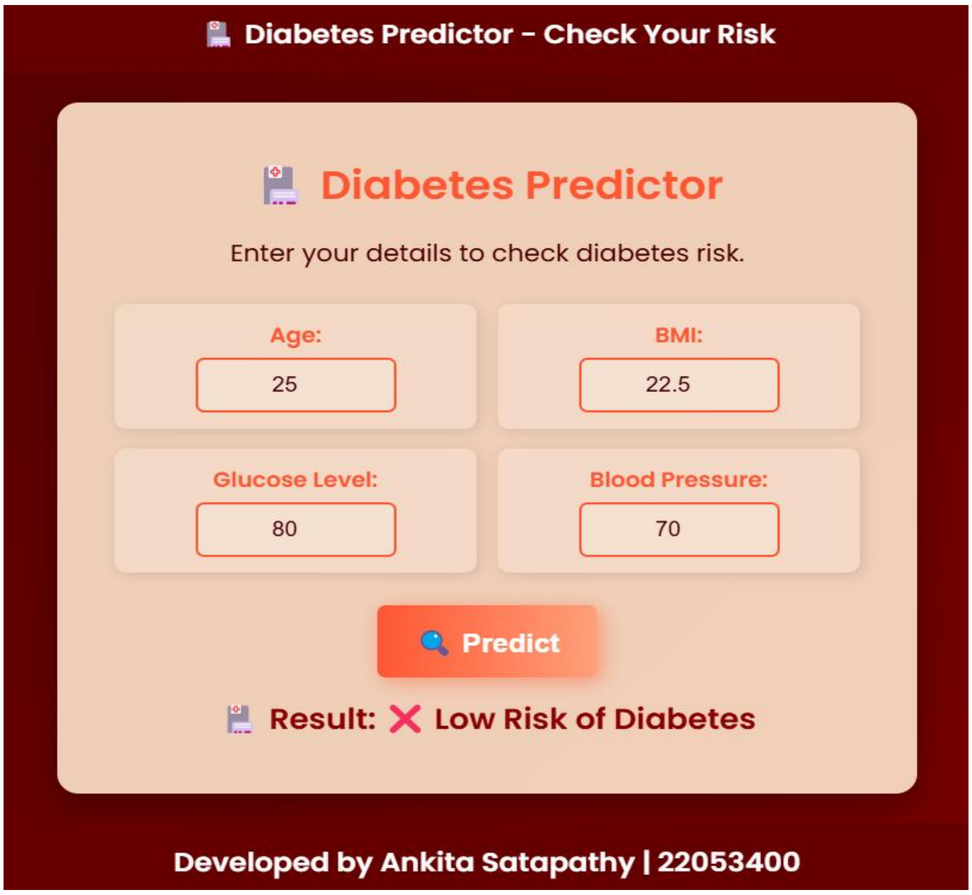


```
async function predict() {
    let features = [];
    let featureIds = ["feature0", "feature1", "feature2", "feature3"];
    for (let id of featureIds) {
        let value = document.getElementById(id).value;
        if (value === "" || isNaN(value)) {
            document.getElementById("result").innerText = "⚠ Enter valid numbers!";
            return;
        }
        features.push(parseFloat(value));
    }
    try {
        let response = await fetch("http://127.0.0.1:8000/predict/", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ features: features })
        });
        let data = await response.json();
        let diabetesStatus = data.prediction === 1 ? "✅ High Risk of Diabetes" : "❌ Low Risk of Diabetes";
        document.getElementById("result").innerText = "🩺 Result: " + diabetesStatus;
    } catch (error) {
        document.getElementById("result").innerText = "❌ Error: " + error.message;
    }
}
```

9. Conclusion

This project successfully implements a custom Decision Tree model for predicting diabetes risk based on selected health parameters. The model was developed using entropy-based splitting and integrates seamlessly with a FastAPI backend. The web interface provides an interactive user experience for real-time predictions.

The front-end design includes a centered UI card with a maroon theme, featuring a clean input layout and a dynamic prediction display. The interactive interface ensures an intuitive user experience for health risk assessment.



💾 Diabetes Predictor - Check Your Risk

💾 Diabetes Predictor

Enter your details to check diabetes risk.

Age: 25

BMI: 22.5

Glucose Level: 80

Blood Pressure: 70

🔍 Predict

💾 Result: ❌ Low Risk of Diabetes

Developed by Ankita Satapathy | 22053400