Project name: Reinforcement Learning in Personalized

Diabetes Management and Treatment Planning

Code:

import numpy as np

class DiabetesEnvironment :

    def \_\_init\_\_(self, max\_glucose\_level):

        self.max\_glucose\_level = max\_glucose\_level

        self.state = 0  # Initial state (glucose level)

    def step(self, action):

        # Take action (e.g., administering insulin)

        # Update state based on action (e.g., glucose level)

        # Return next state, reward, and whether episode is done

        pass

    def reset(self):

        # Reset the environment to initial state

        pass

class QLearningAgent :

    def \_\_init\_\_(self, num\_actions, num\_states, learning\_rate=0.1, discount\_factor=0.9, exploration\_rate=0.1):

        self.num\_actions = num\_actions

        self.num\_states = num\_states

        self.learning\_rate = learning\_rate

        self.discount\_factor = discount\_factor

        self.exploration\_rate = exploration\_rate

        self.q\_table = np.zeros((num\_states, num\_actions))

    def choose\_action(self, state):

        if np.random.uniform(0, 1) < self.exploration\_rate:

            # Explore: choose a random action

            return np.random.choice(self.num\_actions)

        else:

            # Exploit: choose the action with the highest Q-value

            return np.argmax(self.q\_table[state])

    def update\_q\_table(self, state, action, reward, next\_state):

        # Update Q-value using the Q-learning update rule

        old\_q\_value = self.q\_table[state, action]

        td\_target = reward + self.discount\_factor \* np.max(self.q\_table[next\_state])

        new\_q\_value = old\_q\_value + self.learning\_rate \* (td\_target - old\_q\_value)

        self.q\_table[state, action] = new\_q\_value

def main():

    # Initialize environment

    env = DiabetesEnvironment(max\_glucose\_level = ...)

    num\_actions = ...  # Number of actions (e.g., insulin doses)

    num\_states = ...  # Number of states (e.g., glucose levels)

    # Initialize Q-learning agent

    agent = QLearningAgent(num\_actions = num\_actions, num\_states = num\_states)

    num\_episodes = ...  # Number of episodes

    max\_steps\_per\_episode = ...  # Maximum number of steps per episode

    for episode in range(num\_episodes):

        state = env.reset()

        for step in range(max\_steps\_per\_episode):

            # Choose action

            action = agent.choose\_action(state)

            # Take action and observe next state and reward

            next\_state, reward, done = env.step(action)

            # Update Q-table

            agent.update\_q\_table(state, action, reward, next\_state)

            if done:

                break

            state = next\_state

if \_\_name\_\_ == "\_\_main\_\_":

    main()