

|  |  |  |
| --- | --- | --- |
| **Reinforcement Learning** | | |
| Lab Manual | | |
| **Department of Computer Science and Engineering**  **The NorthCap University, Gurugram** | | |
|  | |
|  | |
|  | |
|  | |
|  |  | | |

**Reinforcement Learning**

**Laboratory Manual**

**CSL348**

**Dr. Neetu Singhla**



Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122017, India

Session 2023-24

*Published by:*

**Department of Computer Science & Engineering**

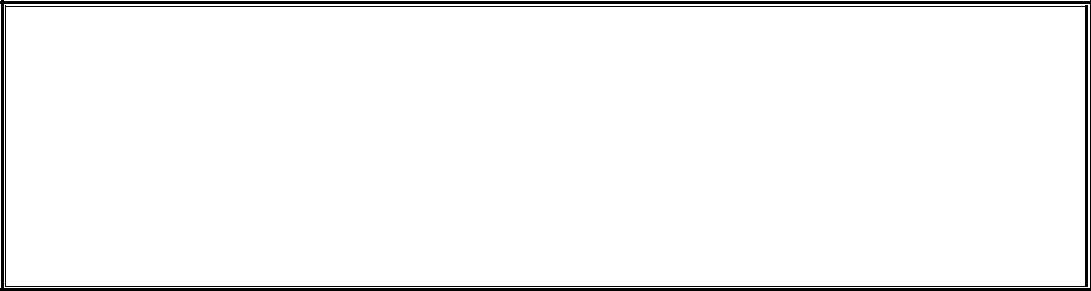
**School of Engineering and Technology**

**The NorthCap University Gurugram**

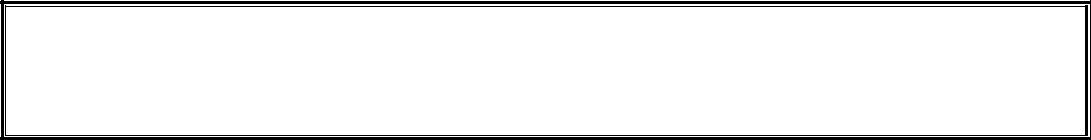
**• Laboratory Manual is for Internal Circulation only**

**© Authors**

*No part of this Practical Record Book may be reproduced, used, stored without prior permission of The NorthCap University*



Copying or facilitating copying of lab work comes under cheating and is considered as use of unfair means. Students indulging in copying or facilitating copying shall be awarded zero marks for that particular experiment. Frequent cases of copying may lead to disciplinary action. Attendance in lab classes is mandatory.



Labs are open up to 7 PM upon request. Students are encouraged to make full use of labs beyond normal lab hours.

**CSL348**

Lab Practical Report



Faculty name : **Dr.Neetu Singhla** Student name:Abhishek Dubey

Roll No.: 21CSU323

Semester: 5th

Group: AIML-B

Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122017, India

Session 2022-2023

**INDEX**

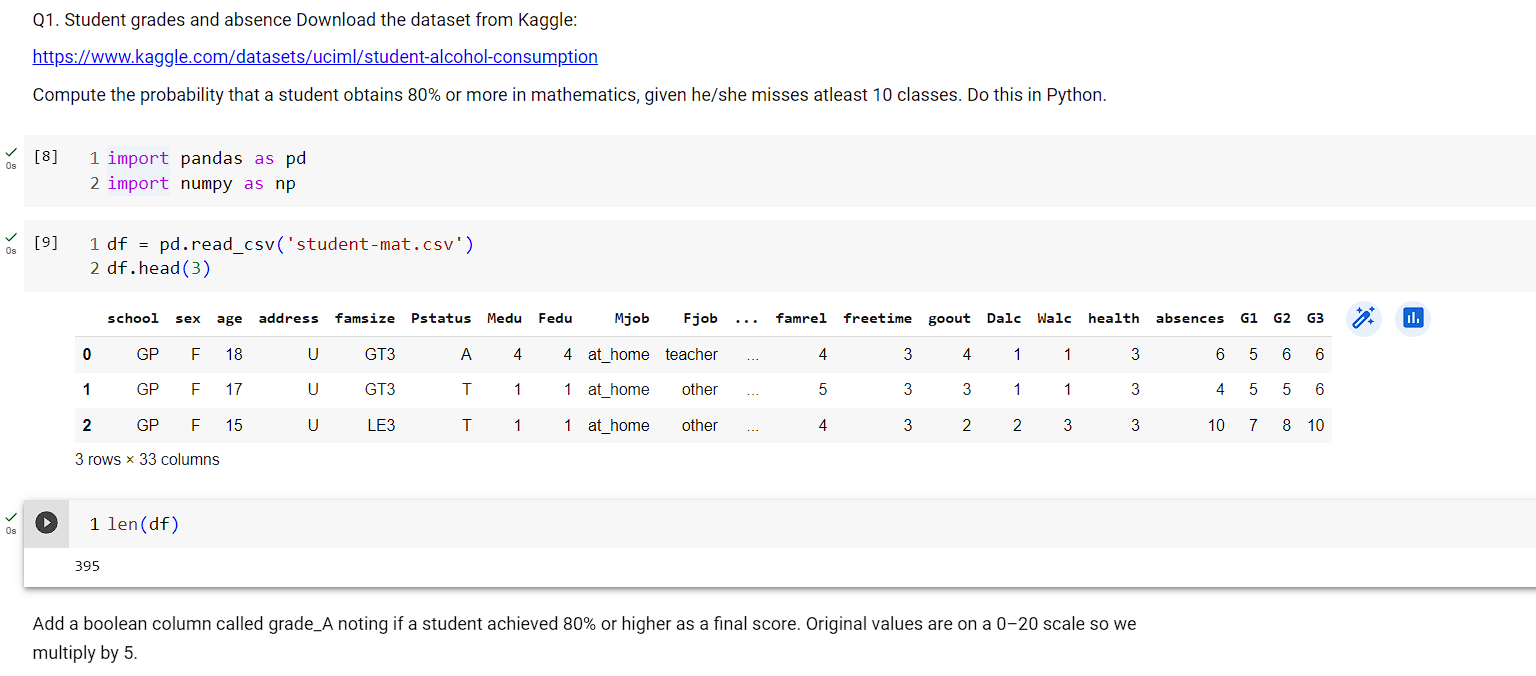
**EXPERIMENT NO. 1**

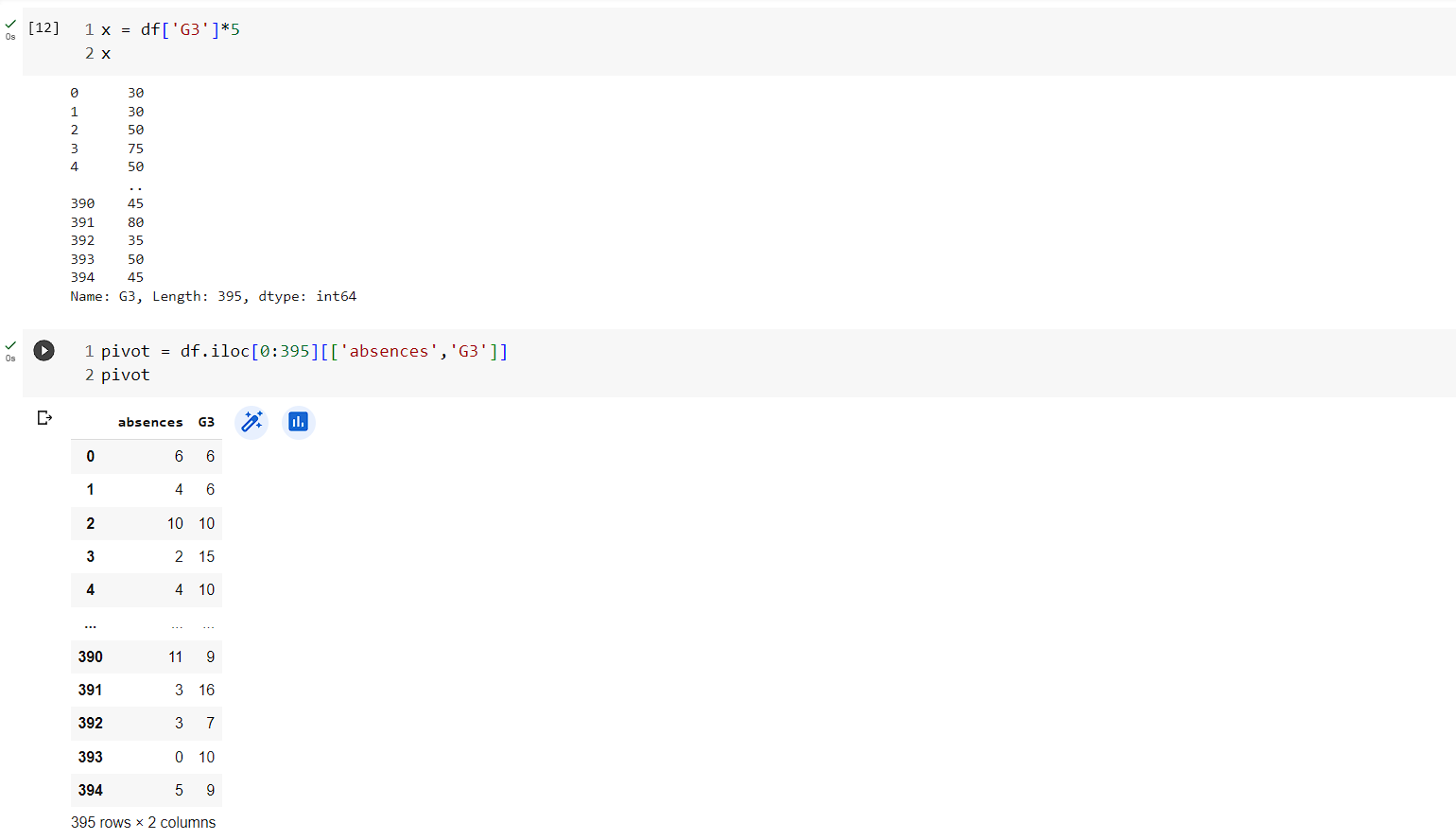
|  |
| --- |
| **Student Name and Roll Number: Abhishek 21CSU323** |
| **Semester /Section:5TH / AIML-B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

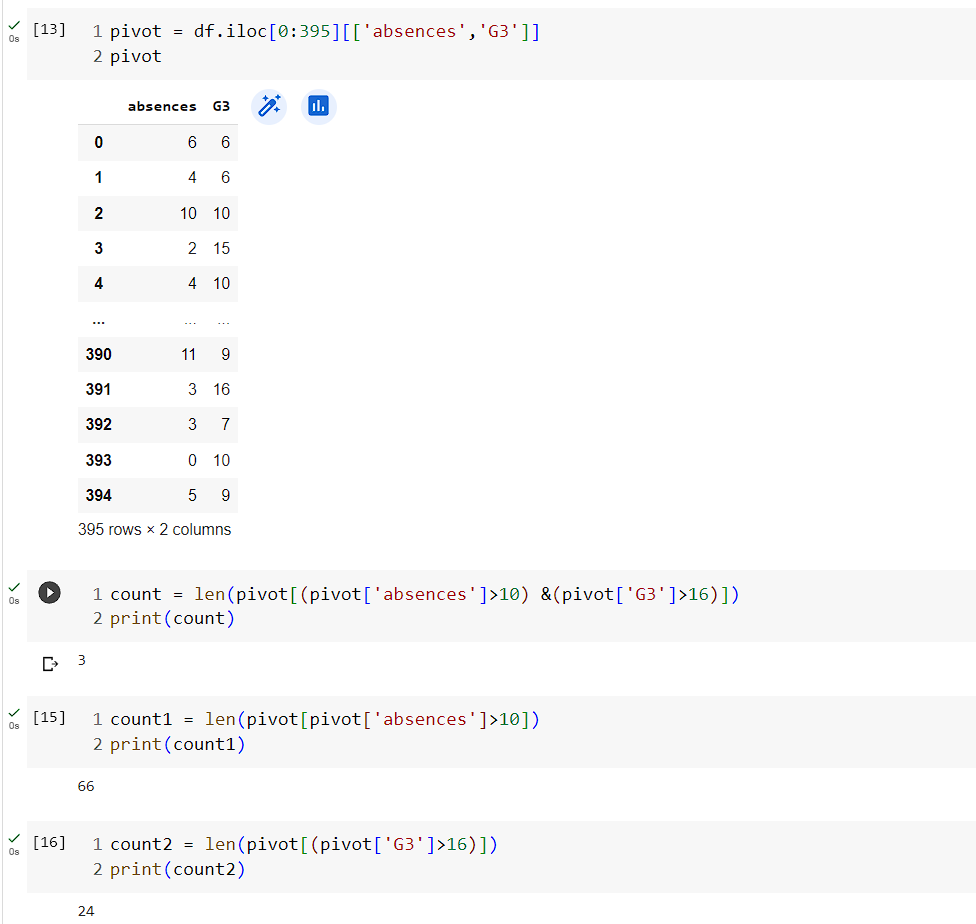
|  |
| --- |
| **Objective(s):**   * Familiarization with probability |
| **Outcome:** Revision of the concepts of probability and probability distributions and implementing the same using Python. |
| **Problem Statement:** Implement Probability using Python |
| **Background Study:**  Python has libraries like Statistics and SciPy. Statistics which contain functions for several descriptive and inferential statistics tasks which can be of help to the students. |
| **Question Bank:**   1. What is difference between discrete and continuous probability distributions?   Discrete distribution is a probability distribution where the random variable can only take on a finite or countable number of values. In contrast, continuous distribution refers to a probability distribution where the random variable can take on any value within a certain range or interval.   1. Enlist some discrete probability distributions.   Discrete Probability Distributions:  Bernoulli Distribution.  Binomial Distribution.  Hyper geometric Distribution.  Negative Binomial Distribution.  Geometric Distribution.  Poisson Distribution.  Multinomial Distribution.   1. Enlist some continuous probability distributions.   Time; a person’s height and weight; |

CODE OUTPUT:

Q1)





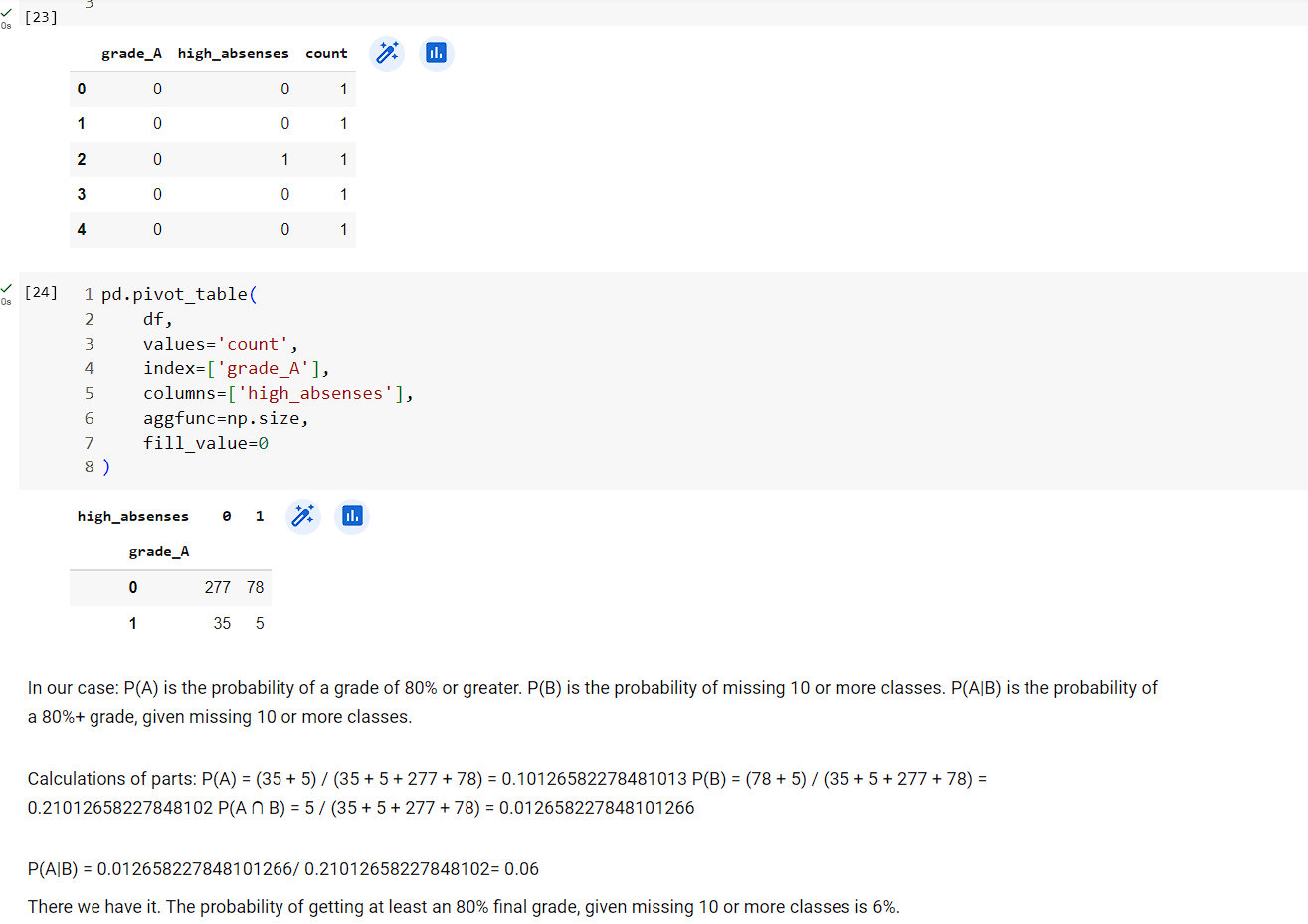






A screenshot of a computer

Description automatically generated



Q2)

A screenshot of a computer

Description automatically generated



Q3)

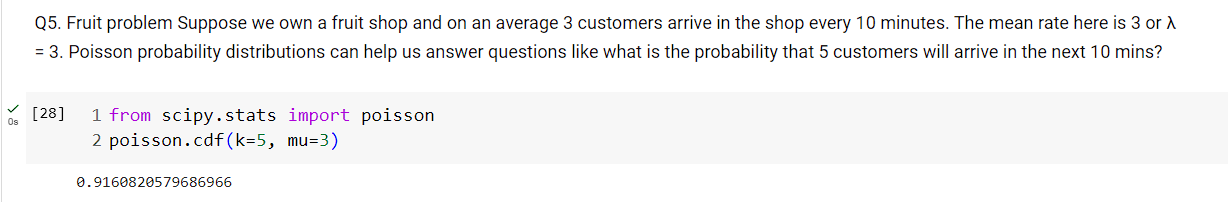
A white background with black text

Description automatically generated

Q4)

A screenshot of a phone

Description automatically generated



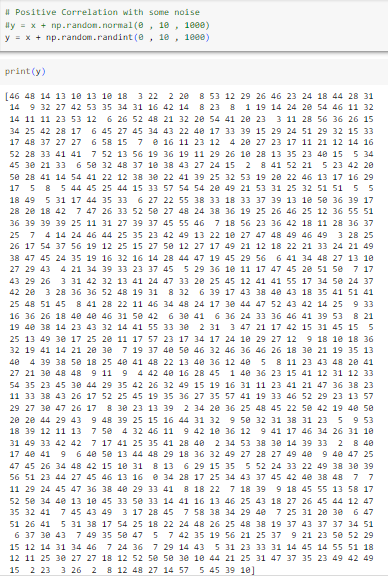
**EXPERIMENT NO. 2**

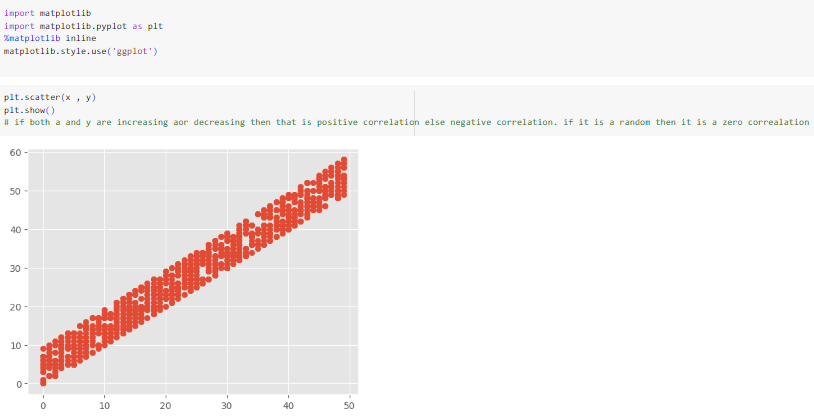
|  |
| --- |
| **Student Name and Roll Number: Abhishek 21CSU323** |
| **Semester /Section:5TH / AIML B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):**  Compute correlation for two given series |
| **Outcome:**Understanding the meaning of correlation |
| **Problem Statement:**Compute Karl Pearson’s and Spearman’s Rank Correlation |
| **Background Study:**In statistics, correlation or dependence is any statistical relationship, whether causal or not, between two random variables or bivariate data. Although in the broadest sense, "correlation" may indicate any type of association, in statistics it normally refers to the degree to which a pair of variables are linearly related. |
| **Question Bank:**   1. Differentiate between correlation and causation.   2. How to compute Spearman’s rank correlation coefficient for repeated ranks.  3. Elucidate on the graphical method for estimating correlation. |

CODE OUTPUT:





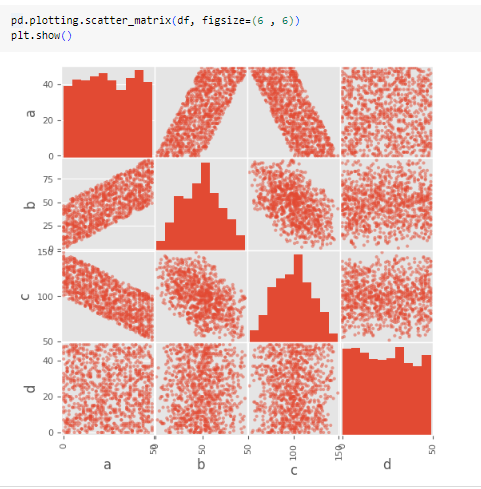


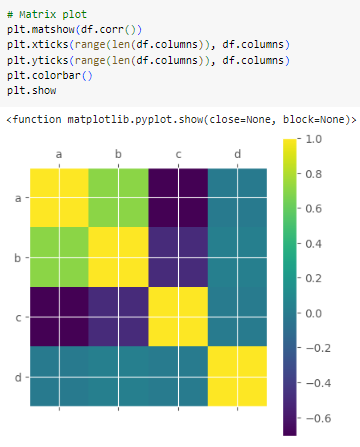
A screenshot of a computer screen

Description automatically generated

A graph with red dots

Description automatically generated





PART 2

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

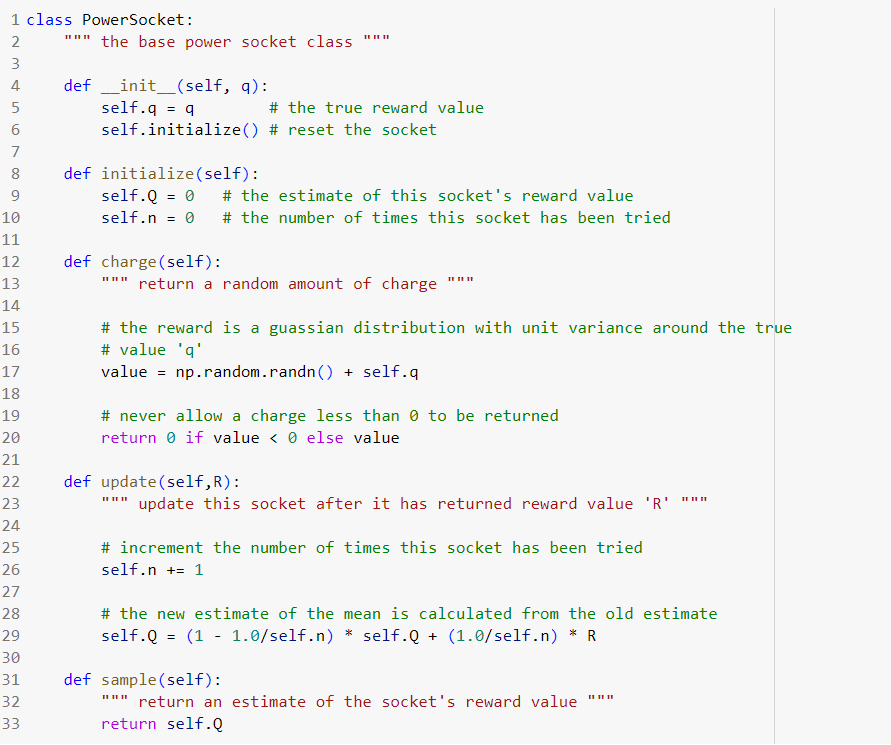
**EXPERIMENT NO. 3**

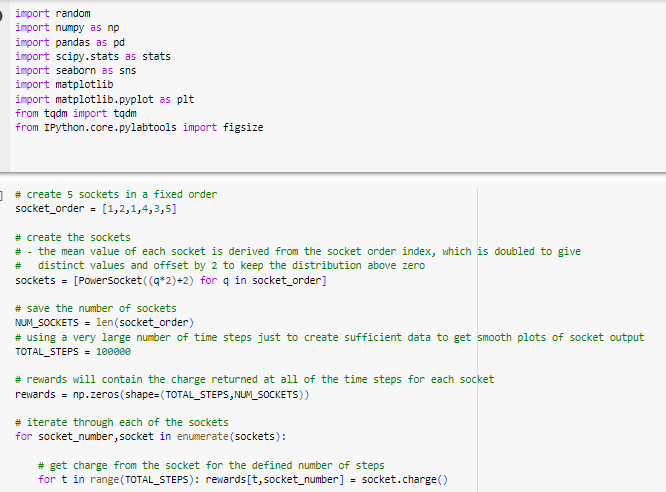
|  |
| --- |
| **Student Name and Roll Number: Abhishek 21CSU323** |
| **Semester /Section:5th / AIML-B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

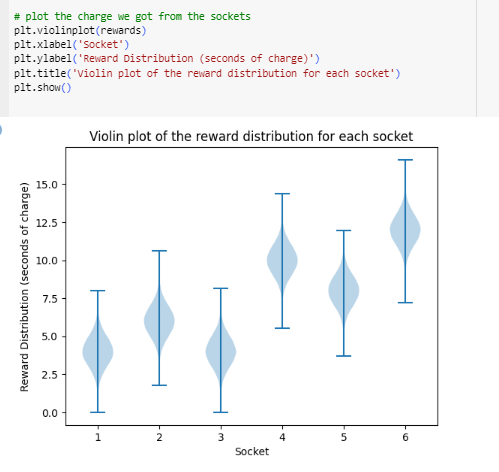
|  |
| --- |
| **Objective(s):**  Solve the Multi-Armed Bandit Problem |
| **Outcome:** Understanding and comparing bandit strategies. |
| **Problem Statement:** Solve the muti-armed bandit problem using the Upper Confidence Bound Algorithm. Compare the reward obtained with random sampling. |
| **Background Study:**In probability theory and machine learning, the **multi-armed bandit problem** (sometimes called the ***K*or *N*-armed bandit problem** is a problem in which a fixed limited set of resources must be allocated between competing (alternative) choices in a way that maximizes their expected gain, when each choice's properties are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources to the choice. This is a classic reinforcement learning problem that exemplifies the exploration–exploitation tradeoff dilemma. |
| **Question Bank:**   1. Differentiate between exploration and exploitation.   Exploration and exploitation are two fundamental strategies in decision-making, especially in contexts where there is uncertainty about the outcomes of different actions. They represent opposing approaches, and striking the right balance between them is crucial for making optimal decisions.   1. Differentiate between greedy and epsilon greedy strategies for solving Multi-armed bandit problem.   The Multi-Armed Bandit problem is a classic exploration-exploitation dilemma where an agent must decide which arm (action) to pull in order to maximize its cumulative rewards over time. Greedy and Epsilon-Greedy are two common strategies used to address this problem. Here's how they differ:  1. \*\*Greedy Strategy\*\*:  - \*\*Exploitation-focused\*\*: The greedy strategy is primarily exploitation-focused. It always selects the action that is currently believed to have the highest expected value (or the highest estimated mean reward).  - \*\*No Exploration\*\*: Greedy strategy does not allocate any effort to explore other arms. It always goes for the action with the highest estimated value, even if it might not be the optimal long-term choice.  - \*\*Risk of Suboptimality\*\*: While the greedy strategy can be efficient in the short term, it can lead to suboptimal long-term performance if it fails to explore and discover higher-rewarding options  \*\*Not Robust to Uncertainty\*\*: Greedy strategy can perform poorly in situations where there is uncertainty or variability in the rewards of different actions.  2. \*\*Epsilon-Greedy Strategy\*\*:  - \*\*Exploration-Exploitation Balance\*\*: Epsilon-Greedy strikes a balance between exploration and exploitation.  - \*\*Probabilistic\*\*: With a probability \( \epsilon \), it chooses an action at random (exploration), and with a probability \(1 - \epsilon\), it selects the action with the highest estimated value (exploitation).  - \*\*Parameter \(\epsilon\)\*\*: The value of \( \epsilon \) is a crucial parameter. A higher \(\epsilon\) encourages more exploration, while a lower \(\epsilon\) leads to more exploitation.  - \*\*Adaptable\*\*: Epsilon-Greedy allows for adaptability to different environments. It can be adjusted to emphasize exploration when uncertainty is high, and exploitation when confidence in estimates is high.  - \*\*Less Risky than Greedy\*\*: Epsilon-Greedy is less risky than the greedy strategy, as it occasionally explores other options even if it believes it has identified the best action.  - \*\*Widely Used\*\*: Epsilon-Greedy is a popular and widely used strategy due to its simplicity and effectiveness in balancing exploration and exploitation.  \*\*Comparison\*\*:  - Greedy strategy tends to be myopic and may get stuck in suboptimal actions if it doesn't explore sufficiently. It's not well-suited for uncertain or changing environments.  - Epsilon-Greedy, on the other hand, provides a controlled way to explore, ensuring that all actions get some attention. This makes it more adaptable to various environments and robust against uncertainty.  In summary, while the Greedy strategy always chooses the current best action, the Epsilon-Greedy strategy incorporates a level of randomness (controlled by the parameter \(\epsilon\)) to explore other actions, striking a balance between exploration and exploitation. This makes Epsilon-Greedy more flexible and robust in uncertain environments.   1. Explain the Upper Confidence Bound Algorithm for solving Multi-armed bandit problem.   The Upper Confidence Bound (UCB) algorithm is a strategy used to solve the Multi-Armed Bandit problem. It aims to balance exploration and exploitation by choosing actions based on upper confidence bounds that estimate the true expected rewards of each arm. The UCB algorithm is particularly effective in scenarios where there is uncertainty about the rewards associated with different actions.  Here's a step-by-step explanation of how the Upper Confidence Bound algorithm works:  1. \*\*Initialization\*\*:  - Initialize estimates of the expected rewards for each arm. This can be done by setting them to some initial value (e.g., 0) or using a random initialization.  2. \*\*Action Selection\*\*:  - For each time step \(t\), select an action to take. Initially, all arms may be considered for selection. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

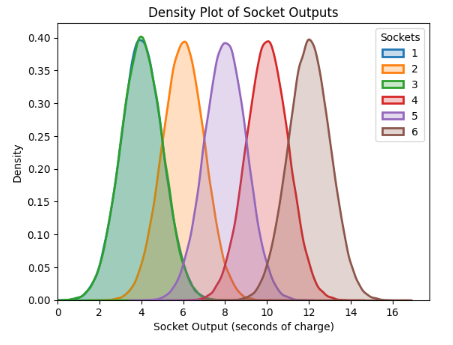
****

****

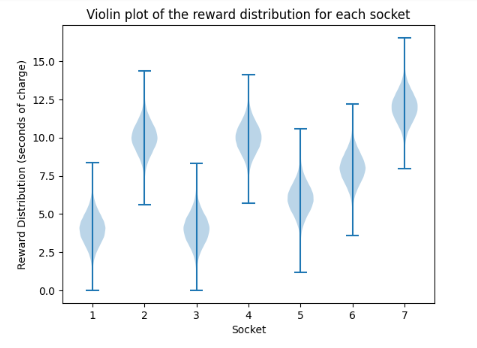
****

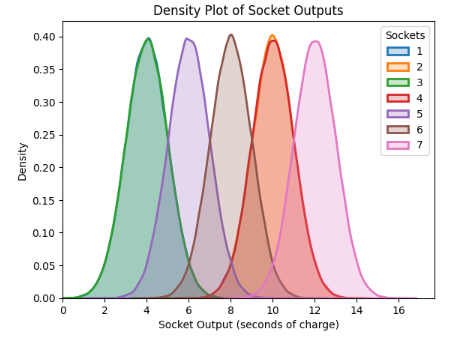
**A screen shot of a computer code

Description automatically generated**

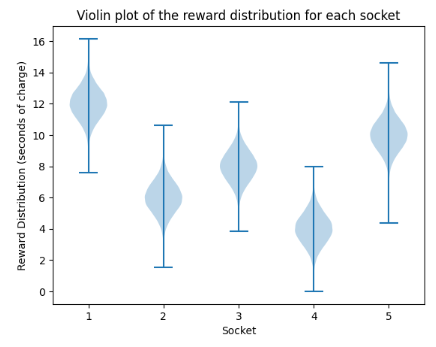
****

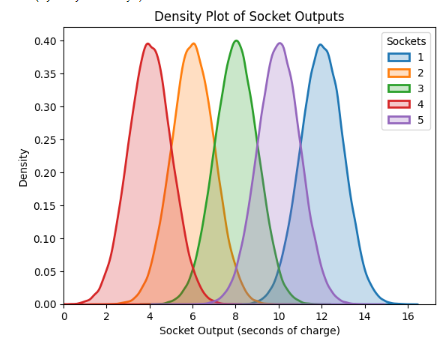
**2nd Socket: [1,4,1,4,2,3,5]**

****

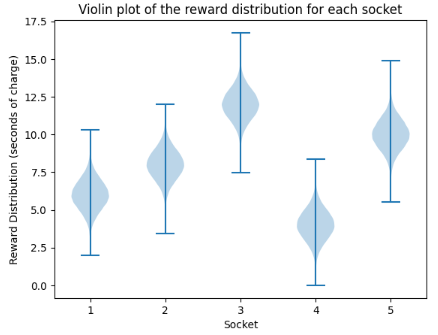
****

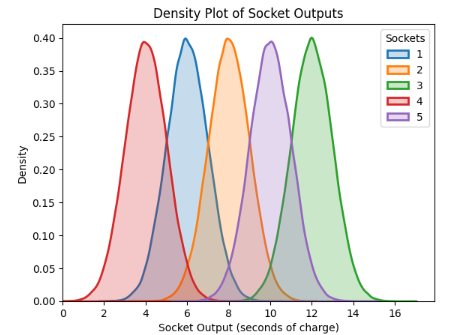
**3rd Socket : [5,2,3,1,4]**

****

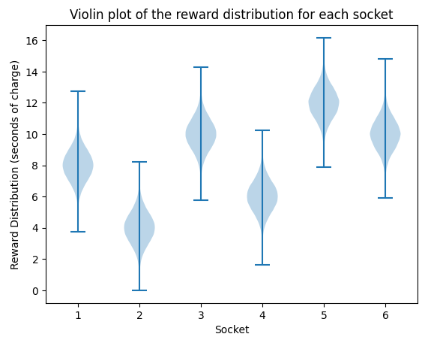
****

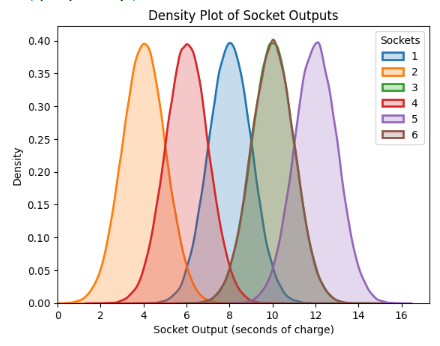
**4th Socket : [2,3,5,1,4]**

****

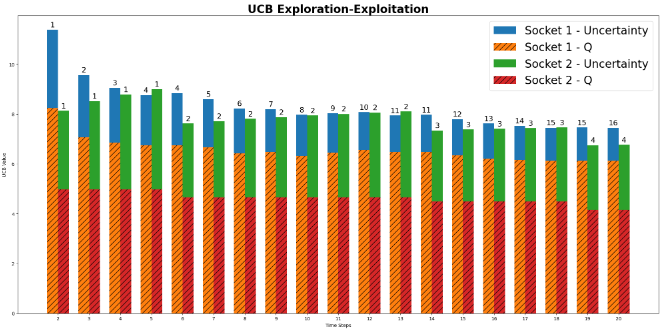
****

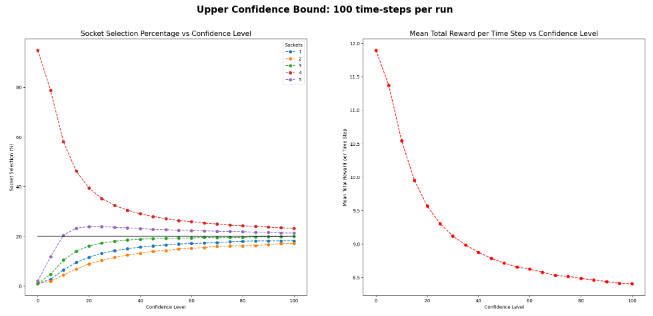
**5th Socket : [3,1,4,2,5,4]**

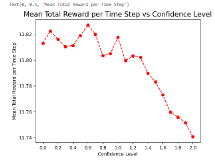
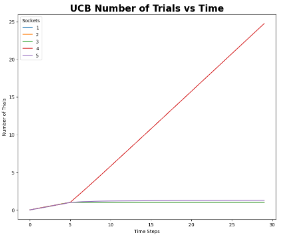
****

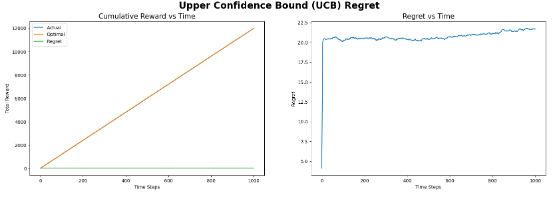
****

**UCB :**

****

****

** **

****

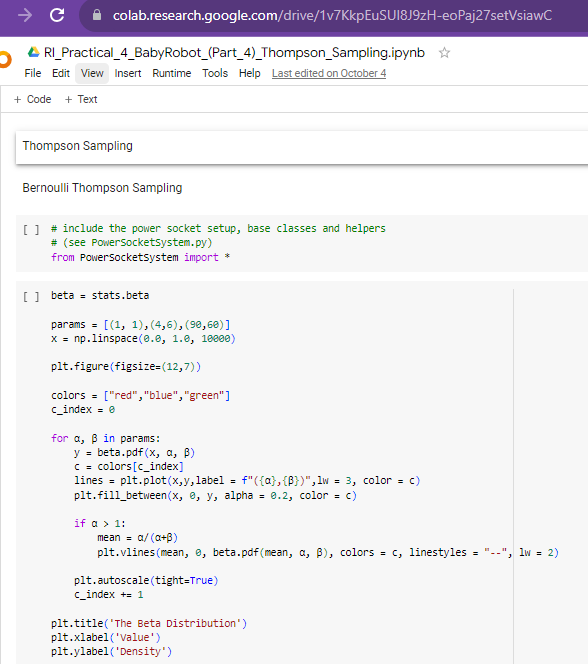
**EXPERIMENT NO. 4**

|  |
| --- |
| **Student Name and Roll Number: Abhishek 21CSU323** |
| **Semester /Section: 5th /AIML – B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):** Solve the Multi-Armed Bandit Problem. |
| **Outcome:** Understand Thompson sampling as a solution to the Multi-Armed Bandit Problem. |
| **Problem Statement:** Write a python program to solve the Multi-Armed Bandit Problem using Thompson Sampling. |
| **Background Study:** Thompson sampling, named after William R. Thompson, is a heuristic for choosing actions that addresses the exploration-exploitation dilemma in the multi-armed bandit problem. It consists of choosing the action that maximizes the expected reward with respect to a randomly drawn belief. |
| **Question Bank:**   1. What are beta distributions and why are they used for Thompson sampling?   Beta distributions are continuous probability distributions defined on the interval [0, 1]. They are parameterized by two positive shape parameters, denoted as α (alpha) and β (beta). The probability density function (PDF) of a beta distribution is given by:  \[f(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}\]  Where:  - \(x\) is a random variable in the interval [0, 1].  - \(\alpha\) and \(\beta\) are the shape parameters.  - \(B(\alpha, \beta)\) is the beta function, which is a normalizing constant to ensure that the PDF integrates to 1.  The mean of a beta distribution is given by \(\frac{\alpha}{\alpha+\beta}\), and its variance is \(\frac{\alpha \beta}{(\alpha + \beta)^2 (\alpha + \beta + 1)}\).  Beta distributions are used in Thompson sampling, a probabilistic approach to solving the exploration-exploitation dilemma in decision-making problems, such as multi-armed bandits.  In Thompson sampling, each arm of a bandit problem is modelled as a probability distribution (often a beta distribution in the case of binary rewards, i.e., success or failure). Initially, the parameters \(\alpha\) and \(\beta\) are set to some prior values that represent the agent's beliefs about the arm's success probability.  After each pull of an arm, the observed outcome (success or failure) is used to update the parameters of the corresponding beta distribution. Specifically, if an arm is pulled and results in a success, \(\alpha\) is incremented by 1; if it results in a failure, \(\beta\) is incremented by 1.  The updated beta distribution then becomes the agent's new belief about the arm's success probability. The agent samples from these distributions and selects the arm with the highest sample, which balances exploration and exploitation.  By using beta distributions, Thompson sampling naturally handles uncertainty in the success probabilities of different arms, making it a powerful strategy for sequential decision-making problems.   1. Compare and contrast Thompson sampling with other bandit strategies.   Thompson Sampling, along with other bandit strategies like Epsilon-Greedy and UCB (Upper Confidence Bound), are algorithms used to solve the exploration-exploitation dilemma in multi-armed bandit problems.   1. Why is Thompson sampling referred to as Bayesian Bandits?   Thompson Sampling is often referred to as "Bayesian Bandits" because of its foundation in Bayesian probability theory. The name stems from the fact that Thompson Sampling employs a Bayesian approach to solving the multi-armed bandit problem.  Here's why it's called "Bayesian Bandits":  1. \*\*Bayesian Framework\*\*: Thompson Sampling operates within a Bayesian framework, which means it makes decisions based on probability distributions that represent uncertainty. In the context of bandits, these distributions (often Beta distributions) are used to model the uncertainty about the success probabilities of each arm.  2. \*\*Prior Beliefs\*\*: Before any actions are taken, the agent starts with prior beliefs about the likelihood of success for each arm. These beliefs are represented by probability distributions.  3. \*\*Updating with Data\*\*: As the agent interacts with the bandit environment and observes outcomes, it updates its beliefs using Bayes' theorem. This means that the prior beliefs are combined with the observed data to form updated (posterior) beliefs.  4. \*\*Sampling from Posterior\*\*: Once the posterior distributions are obtained, Thompson Sampling randomly samples from these distributions to make decisions. This is where the "Sampling" in Thompson Sampling comes from.  5. \*\*Probabilistic Decision-Making\*\*: The agent probabilistically chooses actions based on these samples, which allows it to naturally balance exploration and exploitation.  6. \*\*Adaptive Learning\*\*: The Bayesian framework allows Thompson Sampling to quickly adapt to changes in the environment or the underlying distribution of rewards.  The Bayesian approach in Thompson Sampling is in contrast to other bandit algorithms like Epsilon-Greedy or UCB, which often use deterministic strategies based on point estimates (e.g., mean rewards or upper confidence bounds) without explicitly modeling uncertainty. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

****

**A screen shot of a computer screen

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A computer screen shot of a computer screen

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

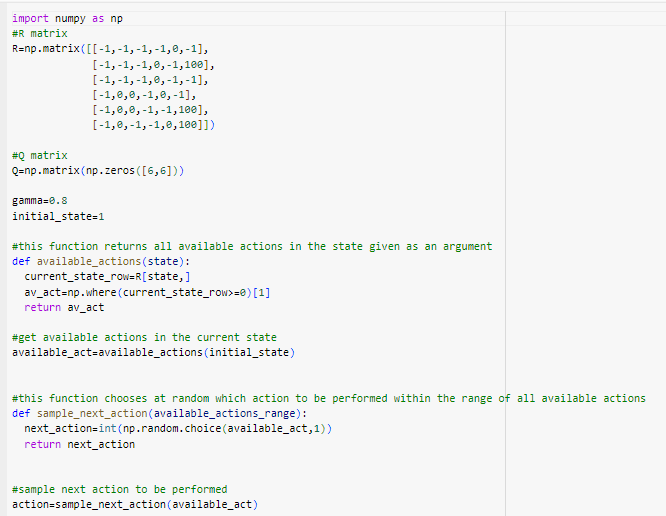
**EXPERIMENT NO. 5**

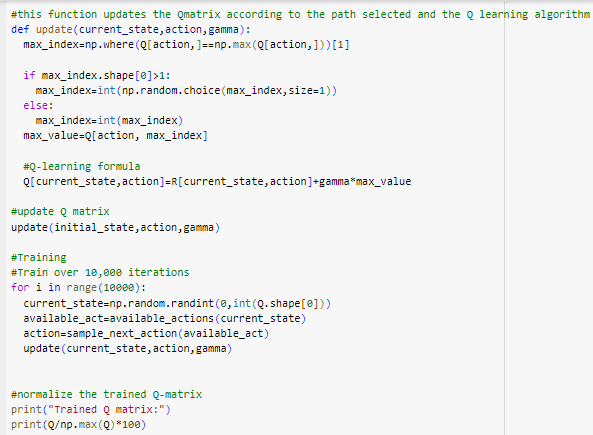
|  |
| --- |
| **Student Name and Roll Number: Abhishek 21CSU323** |
| **Semester /Section: 5th /AIML – B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

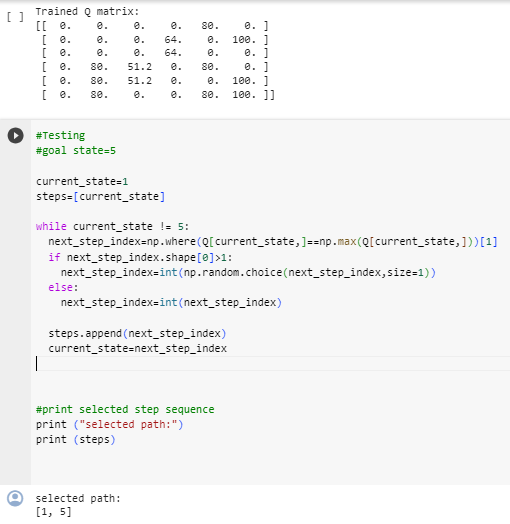
|  |
| --- |
| **Objective(s):** Write python program to implement Q-Learning |
| **Outcome(s):** To understand Q-Learning |
| **Problem Statement:** Implement Q-Learning using Python |
| **Background Study:**Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards without requiring adaptations.  For any finite Markov decision process (FMDP), *Q*-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state. *Q*-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy. "Q" refers to the function that the algorithm computes – the expected rewards for an action taken in a given state. |
| **Question Bank:**   1. Differentiate between policy based and value-based reinforcement learning. 2. What are off-policy and on-policy learners? 3. What is the Bellman equation? 4. What will be the effect(s) of changing the learning rate in Q-Learning? |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

****

****

****

**EXPERIMENT NO. 6**

|  |
| --- |
| **Student Name and Roll Number: Abhishek 21CSU323** |
| **Semester /Section: 5th /AIML – B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):** Understand the Markov Process and Transition Probability Matrix |
| **Outcome(s):** Apply markov process |
| **Problem Statement:** To implement Markov Process in python |
| **Background Study:**A Markov process is a**stochastic process that satisfies the Markov property**  of memorylessness. |
| **Question Bank:**   1. What is the Markov property? 2. What are Markov chains? 3. What are transient, recurring and absorbing states? |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

****

****

****

**EXPERIMENT NO. 7**

|  |
| --- |
| **Student Name and Roll Number: Abhishek 21CSU323** |
| **Semester /Section: 5th / AIML-B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

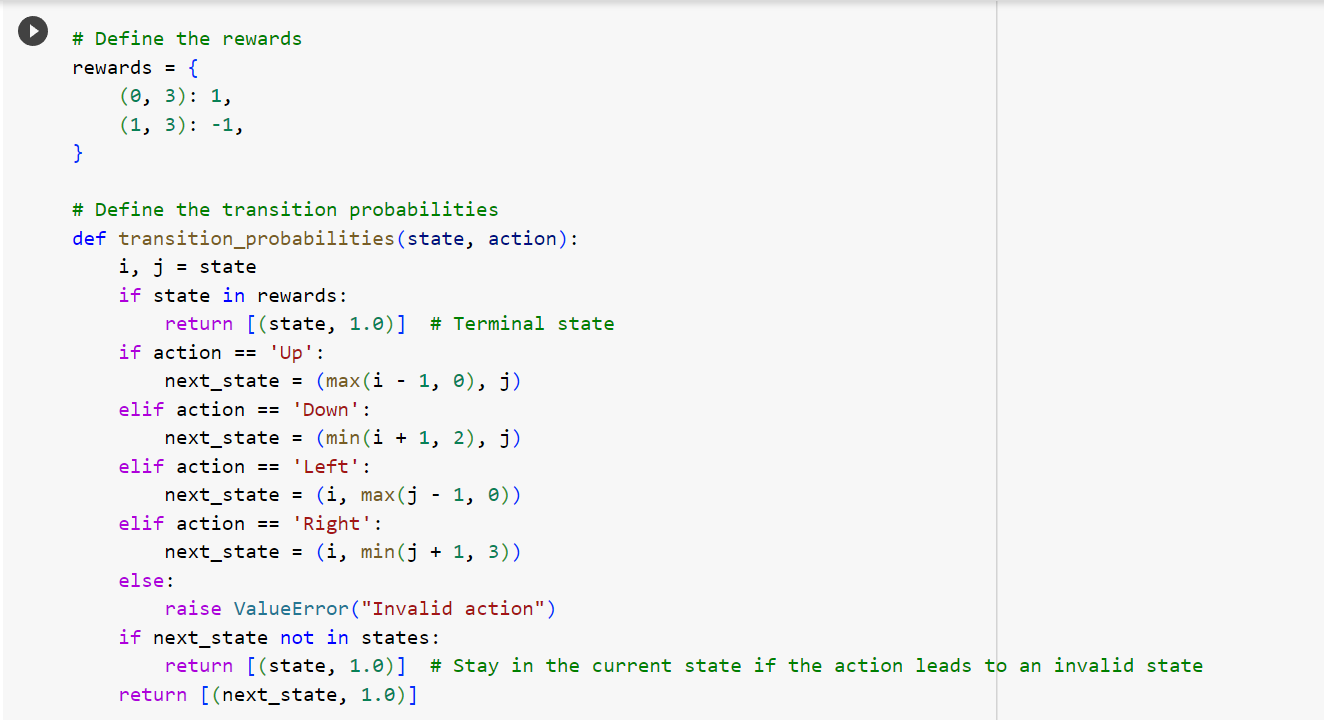
|  |
| --- |
| **Objective(s):** To understand the concept of dynamic programming and policy iteration in RL. |
| **Outcome:**Understand the policy iteration algorithm. |
| **Problem Statement:** Implementation of policy iteration algorithm in dynamic programming. |
| **Background Study:** Policy Iteration is a way to find the optimal policy for given states and actions. **Policy Iteration** takes an initial **policy**, evaluates it, and then uses those values to create an improved **policy**. These steps of evaluation and improvement are then repeated on till convergence. |
| **Question Bank:**   1. What are Bellman expectation and optimality equations?   The Bellman Expectation Equation and the Bellman Optimality Equation are fundamental concepts in reinforcement learning. They describe how the value of a state or state-action pair can be decomposed into immediate rewards and the expected values of future states under a given policy or an optimal policy, respectively.  1. \*\*Bellman Expectation Equation\*\* (also known as the Bellman Expectation Backup):  The Bellman Expectation Equation expresses the value of a state under a policy as the sum of the immediate reward and the expected value of the next state, considering the policy's action selection.  For a state `s` under a policy `π`:  - Value of state `s` (Vπ(s)) is the expected return (cumulative reward) when starting from state `s` and following policy `π`.  - The Bellman Expectation Equation for state `s` is given by:  Vπ(s) = Σ [π(a|s) \* Σ [P(s' | s, a) \* [R(s, a, s') + γ \* Vπ(s')]]]  - `π(a|s)` is the probability of taking action `a` in state `s` under policy `π`.  - `P(s' | s, a)` is the probability of transitioning from state `s` to state `s'` when taking action `a`.  - `R(s, a, s')` is the immediate reward obtained when transitioning from state `s` to state `s'` by taking action `a`.  - `γ` is the discount factor, which represents the importance of future rewards.  This equation provides a way to iteratively update the value of each state under a given policy. It's used in policy evaluation algorithms like the iterative methods (e.g., Policy Iteration, Value Iteration) to estimate the values of states.  2. \*\*Bellman Optimality Equation\*\*:  The Bellman Optimality Equation expresses the value of a state or state-action pair under the optimal policy as the maximum expected return. It defines what it means for a policy to be optimal.  For a state `s`:  - Value of state `s` under the optimal policy (V\*(s)) is the maximum expected return when starting from state `s` and following an optimal policy.  - The Bellman Optimality Equation for state `s` is given by:  V\*(s) = max [Σ [π(a|s) \* Σ [P(s' | s, a) \* [R(s, a, s') + γ \* V\*(s')]]]]  - `max` is taken over all possible policies `π`.  - The rest of the terms have the same meaning as in the Bellman Expectation Equation.  Similarly, for a state-action pair `(s, a)`:  - Value of taking action `a` in state `s` under the optimal policy (Q\*(s, a)) is the maximum expected return when taking action `a` in state `s` and then following an optimal policy.  - The Bellman Optimality Equation for state-action pair `(s, a)` is given by:  Q\*(s, a) = Σ [P(s' | s, a) \* [R(s, a, s') + γ \* max(Q\*(s', a'))]]  - `max` is taken over all possible actions `a'` in state `s'` (the next state).  The Bellman Optimality Equation is used to find the optimal policy and the corresponding optimal value function in reinforcement learning. Solving this equation helps identify the best actions to take in each state to maximize the expected cumulative reward.   1. What is dynamic programming?   Dynamic programming is an optimization technique used to solve problems by breaking them down into smaller overlapping subproblems and efficiently reusing previously computed solutions to those subproblems. It's commonly applied to optimization problems with recursive structures, making it a powerful tool in computer science, algorithms, and operations research.   1. What is a policy?   A policy in reinforcement learning is a strategy that an agent uses to decide its actions in an environment, defining "what to do" based on the current state or state-action pair. It can be deterministic (always chooses the same action) or stochastic (chooses actions with probabilities).   1. Explain the policy evaluation and policy improvement steps in policy iteration.   1. Policy Evaluation:  - Goal: Estimate the value function for the current policy.  - Process: Repeatedly update the value function using the Bellman Expectation Equation until it stabilizes.  2. Policy Improvement:  - Goal: Improve the current policy.  - Process: For each state, select actions that maximize expected returns based on the estimated value function, making the policy "greedy."  Policy iteration alternates between these two steps until the policy becomes optimal, meaning it no longer changes for any state.   1. What do you mean by optimal policy? When is a policy optimal?   An optimal policy in the context of reinforcement learning and Markov Decision Processes (MDPs) is a policy that, when followed by an agent, maximizes the expected cumulative reward over time in the given environment. In other words, it's the best strategy or set of actions an agent can take to achieve the highest possible long-term reward.   1. What is the convergence condition for the policy iteration algorithm?   In Policy Iteration, the convergence condition is met when the current policy remains unchanged during the policy improvement step. This means that for all states, the new policy generated in the iteration is identical to the current policy. When this condition is satisfied, it indicates that the algorithm has found the optimal policy because further iterations would not lead to any policy improvement. In essence, the algorithm has converged to the best possible policy given the current environment and value estimates. |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

**A screenshot of a computer

Description automatically generated**

****

**A white screen with text

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A computer screen shot of a code

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

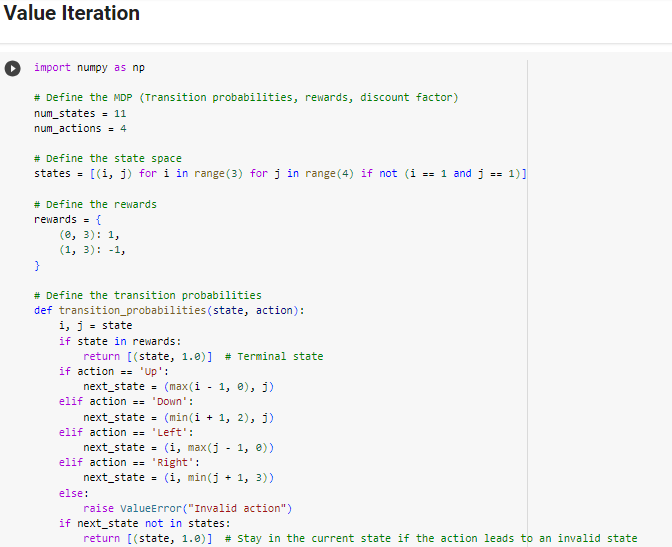
**EXPERIMENT NO. 8**

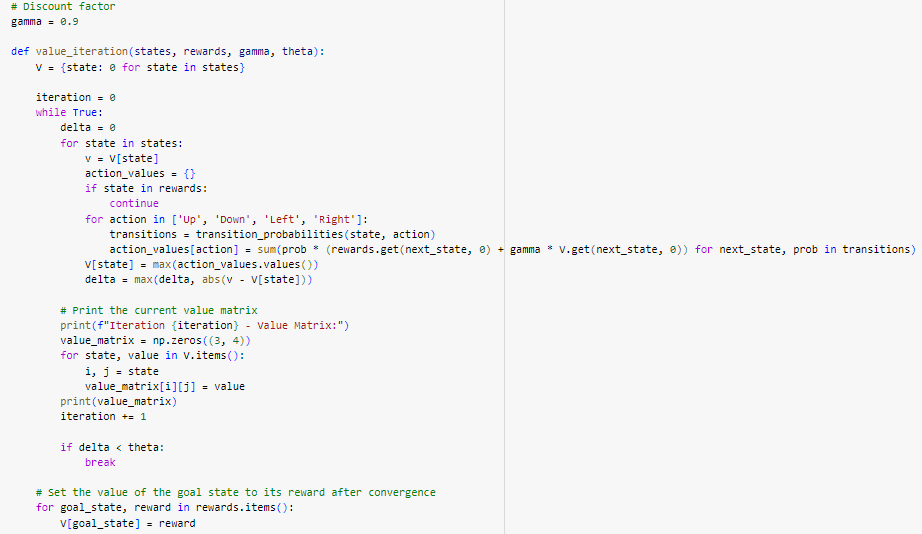
|  |
| --- |
| **Student Name and Roll Number: Abhishek 21CSU323** |
| **Semester /Section: 5th /AIML – B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

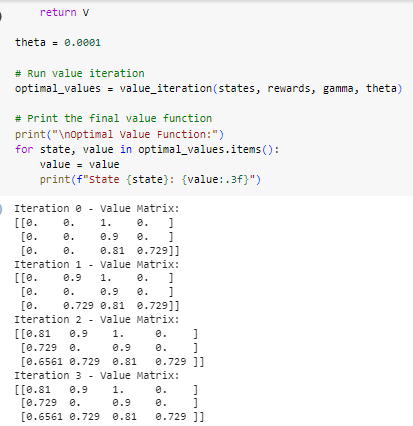
|  |
| --- |
| **Objective(s):** To understand the concepts of dynamic programming and value iteration in RL. |
| **Outcome:**Understand value iteration algorithm. |
| **Problem Statement:** Write a python program to implement value iteration in dynamic programming. |
| **Background Study:** One of the challenges of RL is to find an optimal policy to solve our task. **Value iteration** is a method of computing an optimal policy for an MDP and its**value.In value iteration, we compute the optimal state value function by iteratively updating the state value estimate.** |
| **Question Bank:**  1. What is a Markov Decision Process.  2. Can we obtain the optimal policy using value iteration algorithm?  3. Compare and contrast policy and value iteration algorithms. |

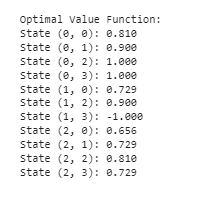
**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

****

****

****

****

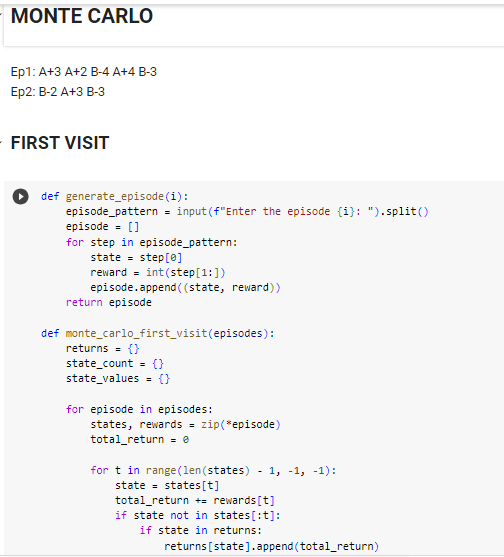
**EXPERIMENT NO. 9**

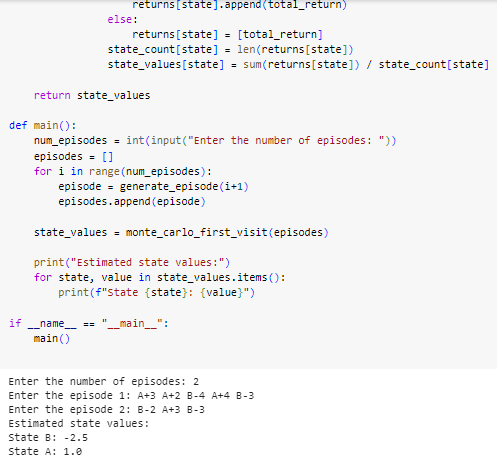
|  |
| --- |
| **Student Name and Roll Number: Abhishek 21CSU323** |
| **Semester /Section: 5th /AIML – B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

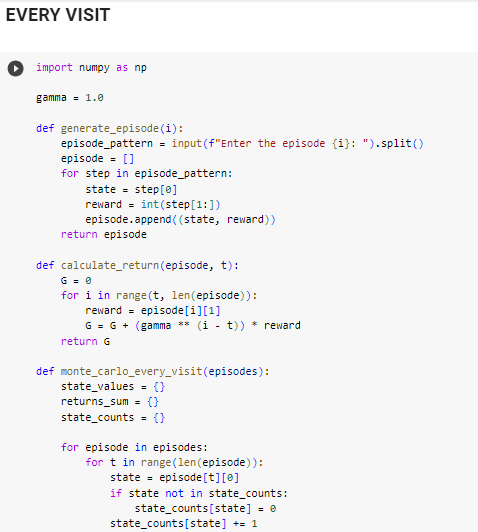
|  |
| --- |
| **Objective(s):** To understand Monte Carlo methods and apply them in Reinforcement Learning scenarios. |
| **Outcome:** Students will be familiarized with Monte Carlo methods. |
| **Problem Statement:** Write Python Program to implement Monte Carlo method to solve the Blackjack problem. |
| **Background Study:** Monte Carlo (MC) methods are a subset of computational algorithms that use the process of repeated random sampling to make numerical estimations of unknown parameters. They allow for the modeling of complex situations where many random variables are involved, and assessing the impact of risk. The uses of MC are incredibly wide-ranging, and have led to a number of ground-breaking discoveries in the fields of physics, game theory, and finance. There are a broad spectrum of Monte Carlo methods, but they all share the commonality that they rely on random number generation to solve deterministic problems.  The Monte Carlo method for reinforcement learning learns directly from episodes of experience without any prior knowledge of MDP transitions. Here, the random component is the return or reward.One caveat is that it can only be applied to episodic MDPs. |
| **Question Bank:**  1. What are episodic MDPs?  2. What are model-free and model-based methods in RL?  3. Differentiate between on-policy and off-policy learning in RL.  4. What are exploring starts in Monte Carlo? |

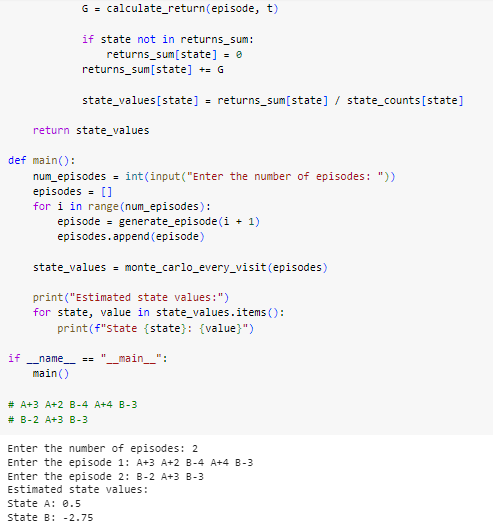
**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

****

****

****

****

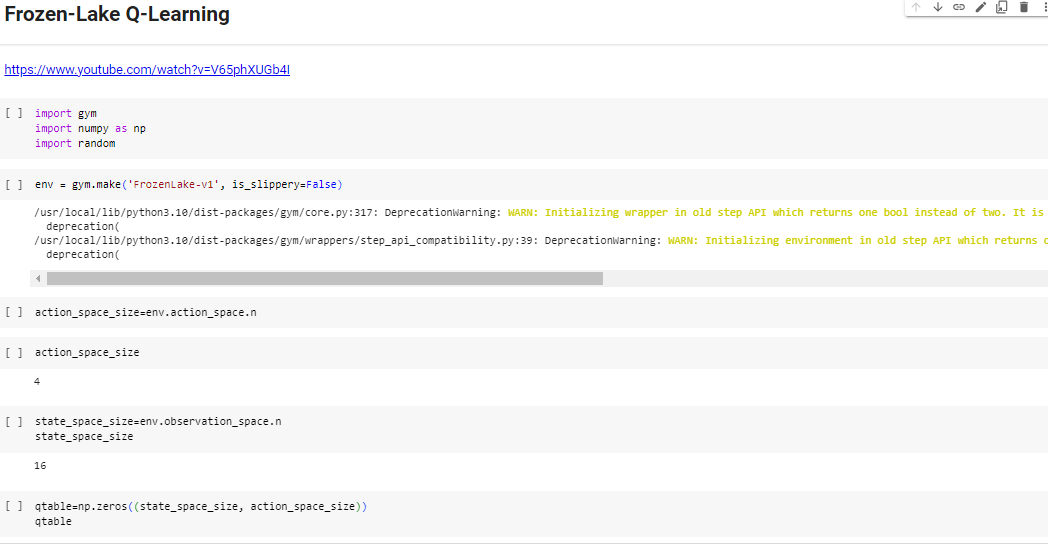
**EXPERIMENT NO. 10**

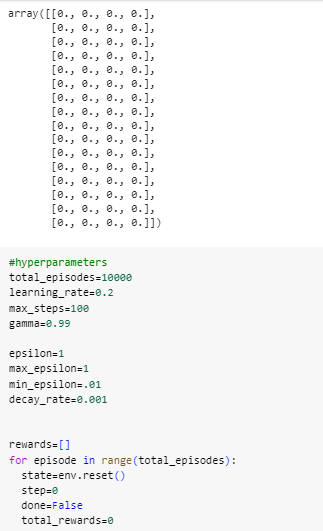
|  |
| --- |
| **Student Name and Roll Number: Abhishek 21CSU323** |
| **Semester /Section: 5th /AIML – B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

|  |
| --- |
| **Objective(s):**   * To Understand Temporal Difference Learning. |
| **Outcome:** Students will understand Model-Free Temporal Difference Learning. |
| **Problem Statement:** Write python code to implement the frozen-lake problem using TD(0). |
| **Background Study:** Temporal difference (TD) learning refers to a class of**model-free reinforcement learning methods which learn by bootstrapping from the current estimate of the value function**. These methods sample from the environment, like Monte Carlo methods, and perform updates based on current estimates, like dynamic programming methods. |
| **Question Bank:**  1.What is bootstrapping?  2. TD methods combine the advantages of both MC and Dynamic programming. Explain.  3. Differentiate between TD(0) and TD(lambda). |

**Student Work Area**

**Algorithm/Flowchart/Code/Sample Outputs**

****

****

****