‘;



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

**Reinforcement Learning**

**Laboratory Manual**

**CSL348**

**Ms. Neetu Singla**



Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122017, India

Session 2023-24

*Published by:*

**Department of Computer Science & Engineering**

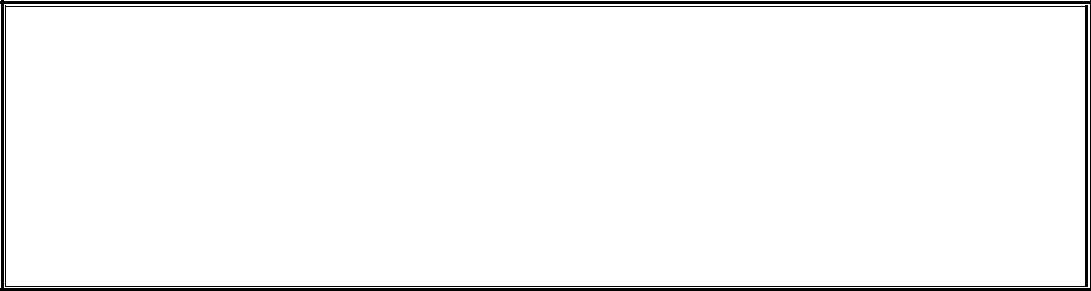
**School of Engineering and Technology**

**The NorthCap University Gurugram**

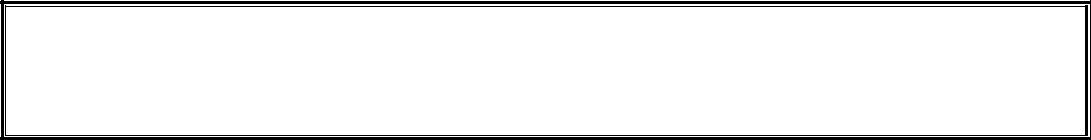
**• Laboratory Manual is for Internal Circulation only**

**© 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.

**PREFACE**

Applied Computational Statistics Laboratory Manual is designed to meet the course and program requirements of NCU curriculum for B.Tech. fourth semester students of CSE branch. The concept of the lab work is to give brief practical experience for basic lab skills to students. It provides the space and scope for self-study so that students can come up with new and creative ideas.

The Lab manual is written on the basis of “teach yourself pattern” and expected that students who come with proper preparation should be able to perform the experiments without any difficulty. A brief introduction to each experiment with information about self-study material is provided. The laboratory exercises will help students to provide a hands-on each exercise that will help them to understand thoroughly. The students are expected to come thoroughly prepared for the lab. General disciplines, safety guidelines and report writing are also discussed.

The lab manual is a part of curriculum for the The NorthCap University, Gurugram. Teacher’s copy of the experimental results and answer for the questions are available as sample guidelines.

We hope that lab manual would be useful to students of CSE branch and author requests the readers to kindly forward their suggestions / constructive criticism for further improvement of the work book.

Author expresses deep gratitude to Members, Governing Body-NCU for encouragement and motivation.

**Authors**

**The NorthCap University**

**Gurugram, India**

|  |  |  |
| --- | --- | --- |
| **CONTENTS** | | |
| S.No. | **Details** | Page No. |
|  |  |  |
| 1 | **Introduction** | vi |
| 2 | **Lab Requirement** | vii |
| 3 | **General Instructions** | viii |
| 4 | **List of Experiments** | x |
| 5 | **List of Flip Experiments** | xi |
| 6 | **List of Projects** | xii |
| 7 | **Rubrics** | xiv |
| 8 | **Annexure 1 (Format of Lab Report)** | xv |
| 9 | **Annexure 2 (Format of Project Report)** | xlviii |

1. **INTRODUCTION**



That ‘learning is a continuous process’ cannot be over emphasized. The theoretical knowledge gained during lecture sessions need to be strengthened through practical experimentation. Thus, practical makes an integral part of a learning process. ­­­­­­­­­­­­­­­­­­­­­

**COURSE OBJECTIVES:**

1. **Understand the basics of descriptive and inferential statistics and be able to apply appropriate descriptive statistical and exploratory methods to analyze datasets.**
2. **Recognize the concept & need of probability in real world. Students will understand the basics of probability, sample space, events, statistics and apply them to real life problems to determine marginal, conditional and joint probabilities.**
3. **Understand the probability mass function and distinguish between the different discrete distributions through application on real-world examples.**
4. **Understand the probability density function and distinguish between the different continuous distributions through application on real-world examples.**
5. **Identify the need for statistical hypothesis testing. Apply the appropriate hypothesis test, interpret the results and devise appropriate strategies.**
6. **Translate real world problems into probability models using Bayesian statistics.**
7. **LAB REQUIREMENTS**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Requirements** | **Details** |
| **1** | **Software Requirements** | Python 3.x, Numpy, Pandas, Matplotlib, Seaborn, statistics, sci-kit learn |
| **2** | **Operating System** | Windows 7 onwards or Linux (32 or 64 bit) |
| **3** | **Hardware Requirements** | 4 GB RAM (Recommended)  2.60 GHz (Recommended) |
| **4** | **Required Bandwidth** | NA |

1. **GENERAL INSTRUCTIONS** 
   1. **General discipline in the lab**
   * Students must turn up in time and contact concerned faculty for the experiment they are supposed to perform.
   * Students will not be allowed to enter late in the lab.
   * Students will not leave the class till the period is over.
   * Students should come prepared for their experiment.
   * Experimental results should be entered in the lab report format and certified/signed by concerned faculty/ lab Instructor.
   * Students must get the connection of the hardware setup verified before switching on the power supply.
   * Students should maintain silence while performing the experiments. If any necessity arises for discussion amongst them, they should discuss with a very low pitch without disturbing the adjacent groups.
   * Violating the above code of conduct may attract disciplinary action.
   * Damaging lab equipment or removing any component from the lab may invite penalties and strict disciplinary action.
   1. **Attendance**

* Attendance in the lab class is compulsory.
* Students should not attend a different lab group/section other than the one assigned at the beginning of the session.
* On account of illness or some family problems, if a student misses his/her lab classes, he/she may be assigned a different group to make up the losses in consultation with the concerned faculty / lab instructor. Or he/she may work in the lab during spare/extra hours to complete the experiment. No attendance will be granted for such case**.**
  1. **Preparation and Performance**
* Students should come to the lab thoroughly prepared on the experiments they are assigned to perform on that day. Brief introduction to each experiment with information about self -study reference is provided on LMS.
* Students must bring the lab report during each practical class with written records of the last experiments performed complete in all respect.
* Each student is required to write a complete report of the experiment he has performed and bring to lab class for evaluation in the next working lab. Sufficient space in work book is provided for independent writing of theory, observation, calculation and conclusion.
* Students should follow the Zero tolerance policy for copying / plagiarism. Zero marks will be awarded if found copied. If caught further, it will lead to disciplinary action.
* Refer **Annexure 1** for Lab Report Format

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Implement Q-Learning and Markov Algorithms with Python and OpenAI | Python  (Jupyter) | 5 | CO3 | 2 hours |

1. **LIST OF FLIP EXPERIMENTS**

|  |  |  |
| --- | --- | --- |
| **Exp. No.** | **Title of the Experiment** | **Mapped CO** |
|  | Apply advanced deep RL algorithms to games  such as Minecraft | CO 1, 2,3,4,5,6 |
|  | Deploy RL algorithms using OpenAI Universe | CO1,2,3,4,5,6 |
|  | Implement basic actor-critic algorithms for  continuous control | CO1,2,3,4,5, 6 |

1. **LIST OF PROJECTS**

|  |  |  |
| --- | --- | --- |
| **Sr No.** | **Project Title** | **Mapped CO** |
|  | Traffic Light Control | CO 1,2,3,4,5,6 |
|  | Robotics | CO1,2,3,4,5,6 |
|  | News Recommendation System. | CO1,2,3,4,5,6 |

1. **RUBRICS (Only for Lab components)**

|  |  |
| --- | --- |
| **Marks Distribution** | |
| **Continuous Evaluation (25 Marks)** | **Project Evaluations (20 Marks)** |
| Each experiment shall be evaluated for 5 marks and at the end of the semester proportional marks shall be awarded out of total 25. | Project shall be evaluated for 20 marks and at the end of the semester viva will be conducted related to the project. |
| **Viva and Reporting (25 Marks)**  Following is the breakup of 25 marks for each  **10 Marks**: Observation & conduct of experiment. Teacher may ask questions about experiment in mid-term viva.  **10 Marks:** Observation & conduct of experiment.  **5 Marks:** For report writing |

**Annexure 1**

**CSL348**

Lab Practical Report



Faculty name: Ms. Neetu Singla Student name: Aditya Sindhu

Roll No.: 21csu278

Semester: 5th

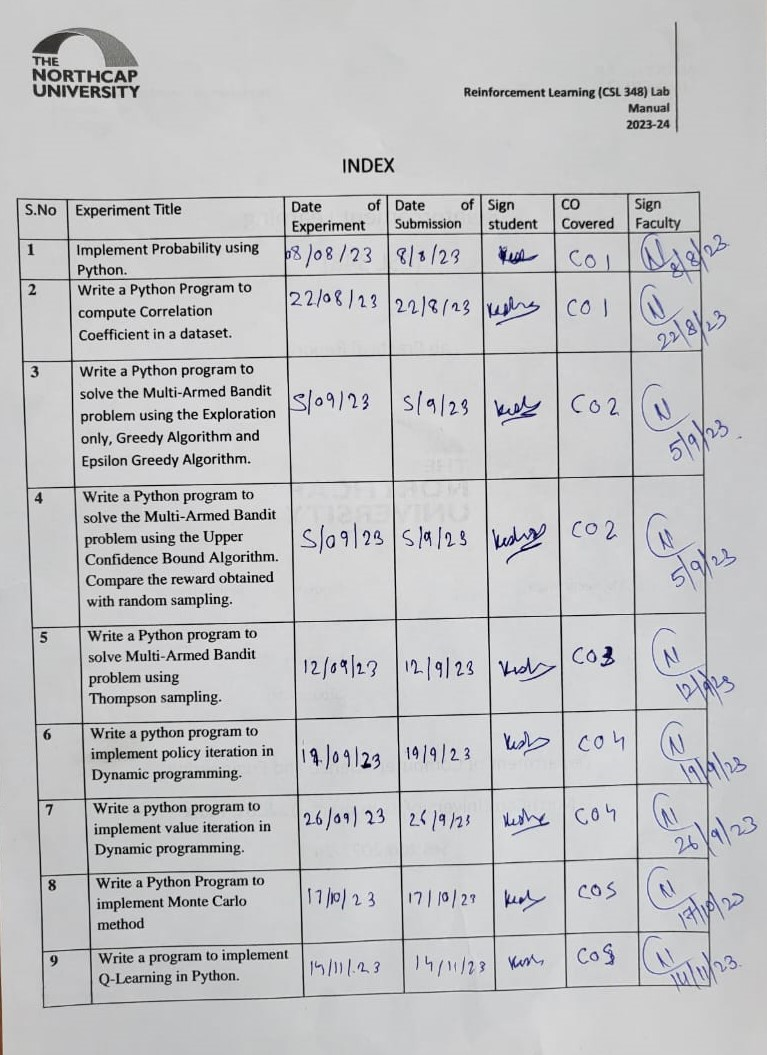
Group: AI-3

Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122017, India

Session 2022-2023

**INDEX**



**EXPERIMENT NO. 1**

|  |
| --- |
| **Student Name and Roll Number: Aditya Sindhu 21csu278** |
| **Semester /Section: 5th/AIML-B** |
| **Link to Code:** |
| **Date: 9/08/23** |
| **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 and continuous probability distributions are two fundamental types of probability distributions that describe the likelihood of different outcomes in various situations. The main difference between them lies in the nature of the outcomes they describe.  **Discrete Probability Distribution:**  In a discrete probability distribution, the random variable can only take on a finite or countable number of distinct values. These values are typically integers or whole numbers. Each possible value is associated with a probability of occurrence. Examples of discrete probability distributions include the binomial distribution, the Poisson distribution, and the geometric distribution. These distributions are used when dealing with situations where outcomes are distinct and separate, such as counting the number of successes in a fixed number of trials or the occurrences of events in a specific interval.  **Continuous Probability Distribution:**  In a continuous probability distribution, the random variable can take on any value within a certain range, often represented by an interval on the real number line. The possible outcomes are not limited to distinct, separate values. Instead, they form a continuum. The probabilities associated with specific individual outcomes are usually infinitesimally small, so probabilities are represented by areas under the probability density function (PDF) curve. Examples of continuous probability distributions include the normal (Gaussian) distribution, the exponential distribution, and the uniform distribution. These distributions are used when dealing with situations involving measurements that can take on any value within a given range, such as time, distance, or temperature.  In summary, the key distinction between discrete and continuous probability distributions is the type of outcomes they describe. Discrete distributions involve distinct, separate values with associated probabilities, while continuous distributions involve a range of values forming a continuum and probabilities are represented as areas under the curve.   1. Enlist some discrete probability distributions.   Here are some common discrete probability distributions:  1.Bernoulli Distribution:Represents a single binary outcome (success or failure) in a single trial of an experiment.  2. Binomial Distribution: Models the number of successes in a fixed number of independent Bernoulli trials.  3. Poisson Distribution: Describes the number of events occurring in a fixed interval of time or space, given a known average rate.  4. Geometric Distribution: Represents the number of trials needed to achieve the first success in a sequence of independent Bernoulli trials.  5. Hypergeometric Distribution: Models the probability of drawing a specific number of successes from a finite population without replacement.  6. Negative Binomial Distribution: Models the number of trials needed to achieve a fixed number of successes in a sequence of independent Bernoulli trials.  7. Discrete Uniform Distribution: Assigns equal probability to a finite number of distinct outcomes.  8. Multinomial Distribution: Generalizes the binomial distribution to situations with more than two outcomes, modeling probabilities of different outcomes in multiple trials.  9. Pascal Distribution: Similar to the negative binomial distribution, representing the number of trials needed to achieve a certain number of successes, but it starts counting from the first success.  10. Zipf Distribution: Describes the distribution of occurrences of different elements in a dataset, often observed in situations with a few dominant elements and many rare ones (e.g., word frequencies in texts).  11. Categorical Distribution: Represents the probabilities of different categories in a categorical variable, often used in situations like modeling preferences or choices.  12. Rademacher Distribution: A discrete distribution where random variables take values +1 or -1 with equal probability, often used in mathematical analysis and signal processing.  13. Logarithmic Distribution: Represents the number of trials needed for the first success in a sequence of Bernoulli trials, where the probability of success decreases geometrically.  14. Yule-Simon Distribution: Models the frequency of observing new and previously seen items in a sequence, commonly used in the study of species distribution and word frequencies   1. Enlist some continuous probability distributions.   Here are some common continuous probability distributions:   1. **Normal (Gaussian) Distribution:** One of the most widely known distributions, it describes a symmetric bell-shaped curve and is commonly used for modeling continuous variables with a tendency to cluster around a mean value. 2. **Uniform Distribution:** Represents outcomes that are equally likely within a certain range, resulting in a constant probability density over that range. 3. **Exponential Distribution:** Models the time between events in a Poisson process, often used to describe the time between arrivals of events in a queue or system. 4. **Gamma Distribution:** Generalizes the exponential distribution and is often used to model waiting times or durations when multiple exponential distributions are combined. 5. **Beta Distribution:** Represents the distribution of probabilities in the interval [0, 1], often used for modeling proportions or probabilities. 6. **Chi-Square Distribution:** Used in hypothesis testing and statistical inference, it's the distribution of the sum of squared standard normal random variables. 7. **Cauchy Distribution:** Has heavy tails and lacks a defined mean or variance, often used in situations where extreme values are possible. 8. **Weibull Distribution:** Commonly used to model failure rates of systems, representing the distribution of time to failure. 9. **Log-Normal Distribution:** Describes variables that are the result of the exponential of normally distributed values, often used to model quantities that cannot be negative, like stock prices. 10. **Pareto Distribution:** Describes distributions with a heavy tail, often used to model distributions of wealth, income, or other phenomena with a few large values and many small values. 11. **Laplace Distribution:** Also known as the double-exponential distribution, it's used for modeling data with heavy tails and a center peak. 12. **Normal Inverse Gaussian (NIG) Distribution:** Has both heavy tails and asymmetry, making it suitable for modeling financial data and extreme events. 13. **Logistic Distribution:** S-shaped distribution used for modeling growth, often used in biology, economics, and other fields. 14. **Triangular Distribution:** Has a triangular shape and is often used when the true distribution of a variable is unknown or when only limited data is available. 15. **Gumbel Distribution:** Often used to model extreme value distributions, such as the maximum value in a sample over a period of time. |

**Student Work Area**

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

**A screenshot of a computer

Description automatically generatedA screenshot of a computer code

Description automatically generatedA screenshot of a computer

Description automatically generated**

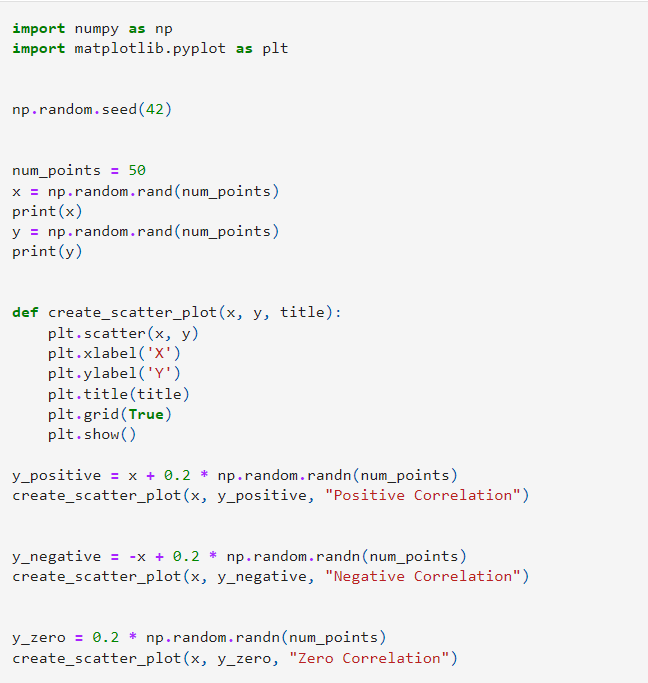
**EXPERIMENT NO. 2**

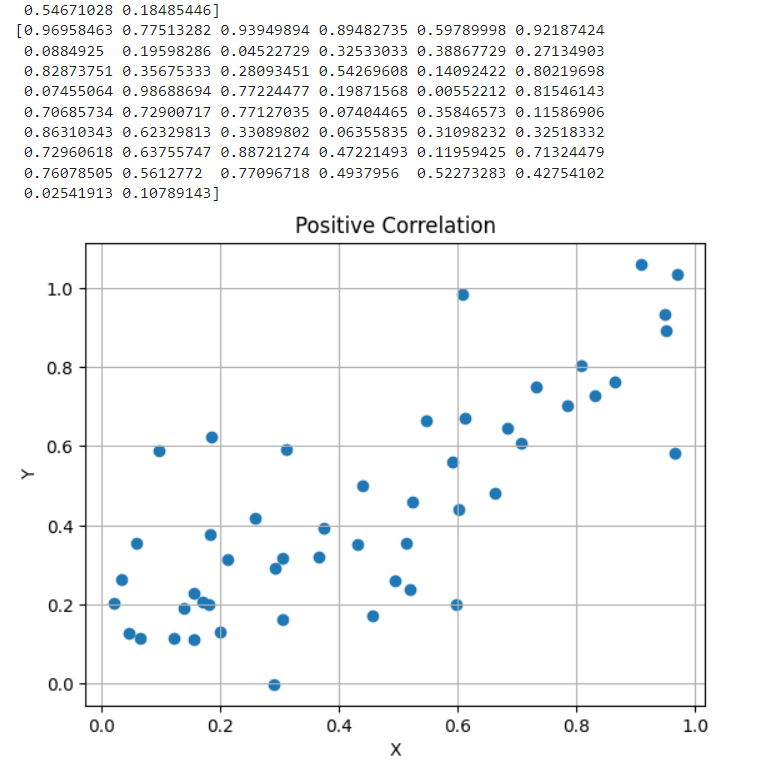
|  |
| --- |
| **Student Name and Roll Number: Aditya Sindhu 21csu278** |
| **Semester /Section: 5th/AIML-B** |
| **Link to Code:** |
| **Date: 16/08/23** |
| **Faculty Signature:** |
| **Marks/Grade:** |

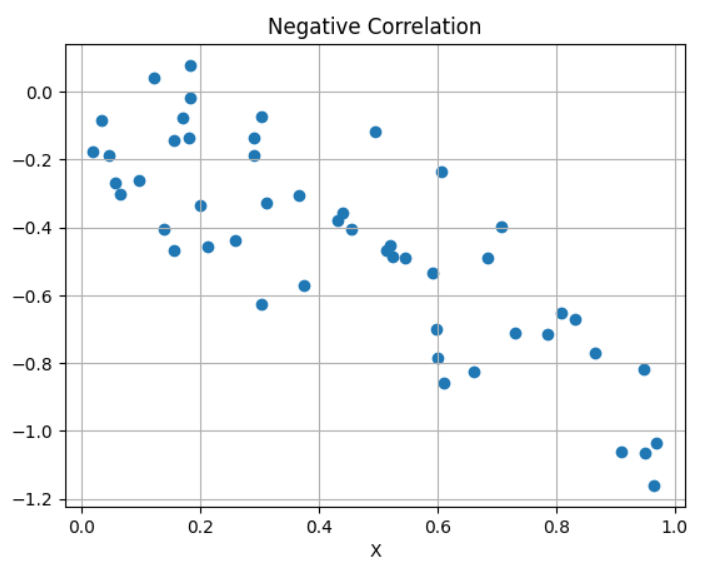
|  |
| --- |
| **Objective(s):**  Compute correlation for two given series |
| **Outcome:** Understanding the meaning of correlation |
| **Problem Statement:** Compute Karl Pearson’s and Spearman’s Rank Correlation |
| **Background Study:** In statistics, correlation or dependence is any statistical 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. |

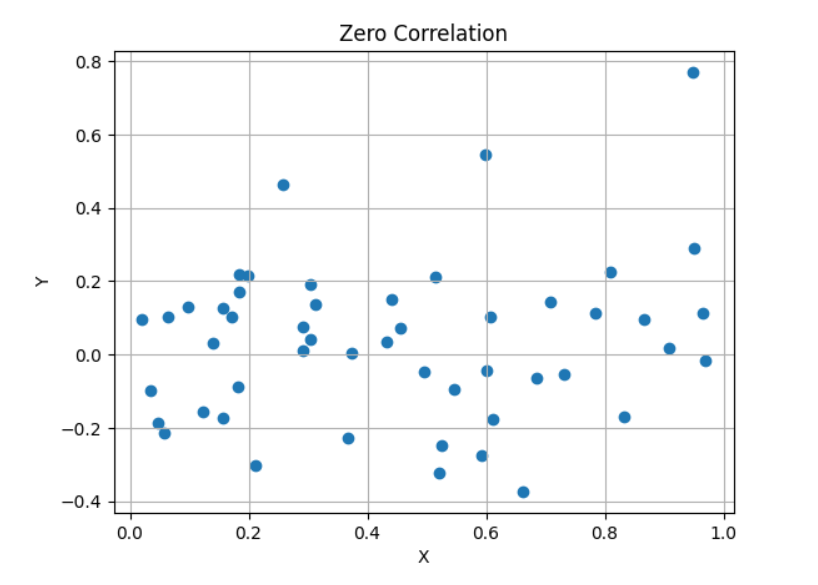
**Student Work Area**

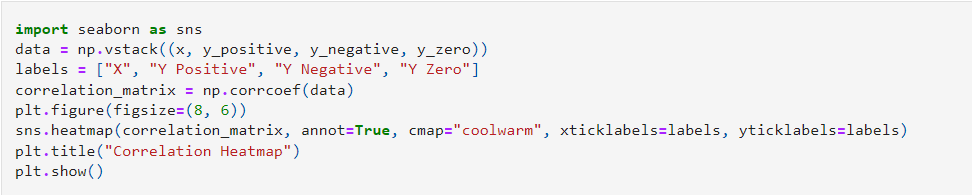
**Algorithm/Flowchart/Code/Sample Outputs**

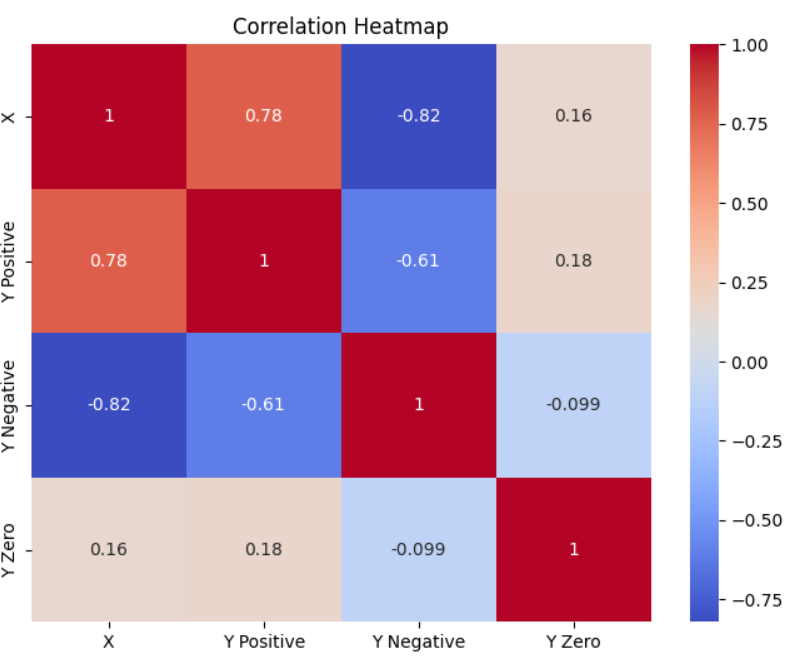
****

****

****

****

****

****

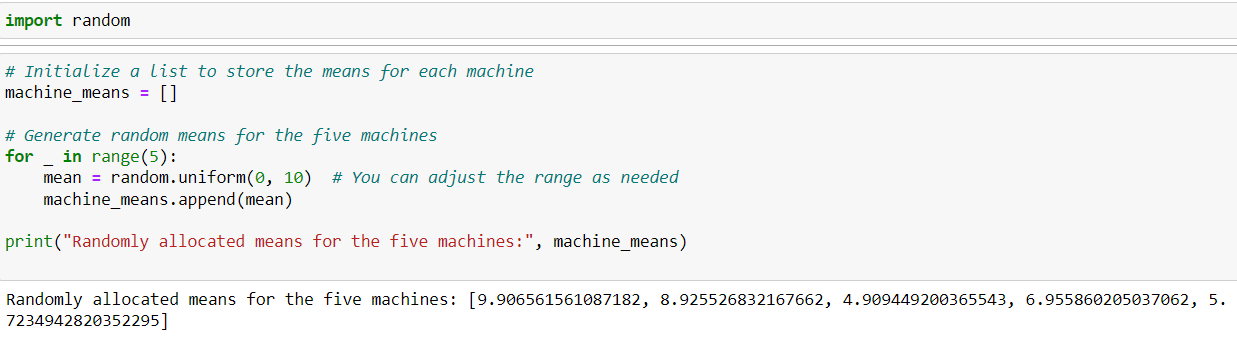
**EXPERIMENT NO. 3**

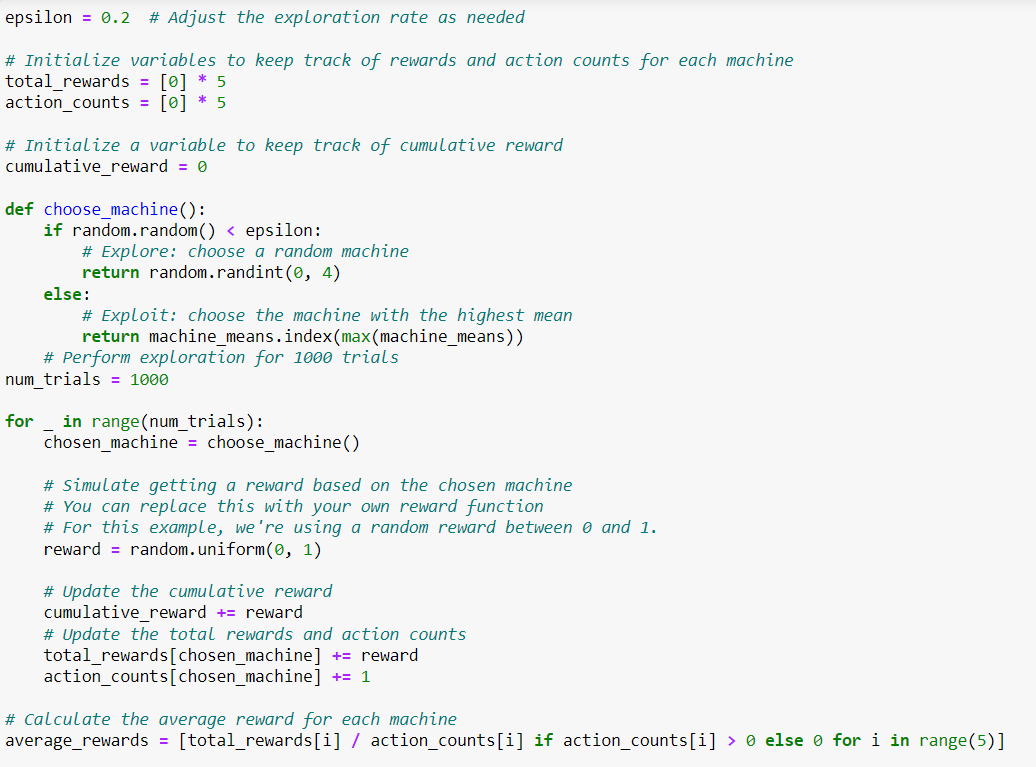
|  |
| --- |
| **Student Name and Roll Number: Aditya Sindhu 21csu278** |
| **Semester /Section:5th/AIML-B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

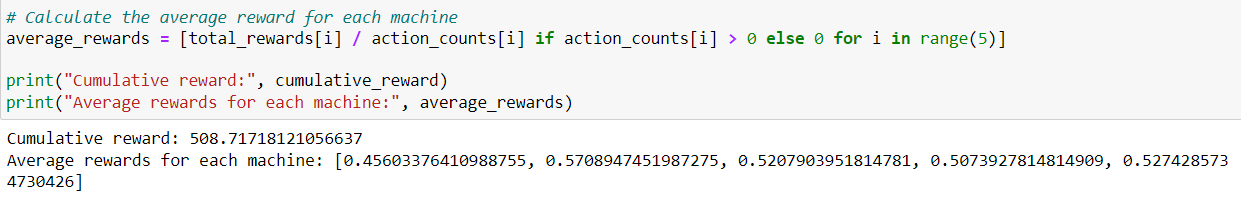
|  |
| --- |
| **Objective(s):**  Solve the Multi-Armed Bandit Problem |
| **Outcome:** Understanding and comparing bandit strategies. |
| **Problem Statement:** Solve the muti-armed bandit problem using the exploration only, greedy algorithm and epsilon greedy. |
| **Background Study:** In probability 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. \*\*Exploration\*\*:  - \*\*Definition\*\*: Exploration involves trying out different actions or options to gather information about their potential outcomes.  - \*\*Purpose\*\*: The primary goal of exploration is to learn more about the environment, uncovering the uncertainties and potential benefits of different choices.  - \*\*Risk Tolerance\*\*: It often involves taking risks, as the agent might choose actions that are not currently believed to be the best, but have the potential to yield valuable information.  - \*\*Long-term Gain\*\*: While exploration may not immediately lead to the best outcomes, it contributes to long-term learning and better decision-making in the future.  - \*\*Promotes Adaptation\*\*: Exploration is essential for adapting to changing environments and for discovering new, possibly more rewarding, options.  2. \*\*Exploitation\*\*:  - \*\*Definition\*\*: Exploitation involves choosing actions that are currently believed to be the best based on existing knowledge or estimates.  - \*\*Purpose\*\*: The main objective of exploitation is to maximize immediate rewards or benefits, capitalizing on the current knowledge or estimates of action values.  - \*\*Risk Aversion\*\*: It typically involves less risk, as the agent leans towards actions that are already perceived as high-performing.  - \*\*Short-term Gain\*\*: Exploitation often leads to maximizing short-term gains, making the most of current knowledge and beliefs.  - \*\*May Stagnate Learning\*\*: Overemphasis on exploitation without sufficient exploration can lead to stagnation in learning and missed opportunities for discovering better options.  \*\*Balancing Exploration and Exploitation\*\*:  - In many decision-making scenarios, finding the right balance between exploration and exploitation is critical for achieving optimal results.  - Too much exploration can lead to inefficient decision-making, while too much exploitation can result in suboptimal long-term performance.  \*\*Example\*\*:  Consider a scenario of selecting a restaurant for dinner.  - \*\*Exploration\*\*: Trying out a new restaurant you've never been to before is exploration. You might discover a hidden gem, but there's also a risk it might not meet your expectations.  - \*\*Exploitation\*\*: Going to your favorite restaurant because you know you enjoy their food is exploitation. It's a safe choice based on your prior positive experiences.  In complex decision-making environments, finding the right balance between exploration and exploitation is often a key challenge, and different algorithms and strategies (such as Thompson Sampling, Epsilon-Greedy, and UCB) are designed to address this dilemma in various contexts.   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.    - \*\*Deterministic\*\*: It deterministically chooses the action with the highest estimated value.  - \*\*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.  3. \*\*Upper Confidence Bound Calculation\*\*:  - For each arm \(i\), calculate the Upper Confidence Bound (UCB) using the following formula:  \[UCB\_i(t) = \hat{Q}\_i(t) + c \sqrt{\frac{\ln(t)}{N\_i(t)}}\]  - \(\hat{Q}\_i(t)\) is the estimated expected reward of arm \(i\) at time \(t\).  - \(N\_i(t)\) is the number of times arm \(i\) has been selected up to time \(t\).  - \(c\) is a user-defined parameter that controls the balance between exploration and exploitation. A higher \(c\) encourages more exploration.  4. \*\*Action Selection based on UCB\*\*:  - Select the arm with the highest UCB value at time \(t\):  \[A\_t = \arg \max\_i UCB\_i(t)\]  5. \*\*Execute Action and Observe Reward\*\*:  - Pull the selected arm, receive a reward, and observe the outcome.  6. \*\*Update Estimates\*\*:  - Update the estimated expected reward \(\hat{Q}\_i(t)\) of the chosen arm based on the observed reward:  \[\hat{Q}\_i(t+1) = \frac{\sum\_{j=1}^{t} R\_j \cdot \mathbb{1}(A\_j=i)}{N\_i(t) + 1}\]  - \(R\_j\) is the reward obtained at time step \(j\).  - \(\mathbb{1}(A\_j=i)\) is an indicator function that equals 1 if arm \(i\) was chosen at time step \(j\), and 0 otherwise.  - \(N\_i(t) + 1\) is the updated count of times arm \(i\) has been selected.  7. \*\*Repeat\*\*:  - Repeat steps 3-6 for subsequent time steps.  \*\*Key Points\*\*:  - The UCB algorithm uses confidence bounds to balance exploration and exploitation. The term \(\sqrt{\frac{\ln(t)}{N\_i(t)}}\) represents the confidence interval, which widens as the number of samples \(N\_i(t)\) for arm \(i\) increases.  - The parameter \(c\) controls the level of exploration. A higher \(c\) encourages more exploration, while a lower \(c\) favors exploitation.  - UCB is effective in environments with uncertainty about the rewards of different actions. It is more robust than a purely greedy strategy.  - However, UCB may be overly optimistic in certain situations, potentially leading to over-exploitation.  Overall, the UCB algorithm provides a principled approach to the exploration-exploitation dilemma in the multi-armed bandit problem, making it a powerful strategy in a wide range of applications. |

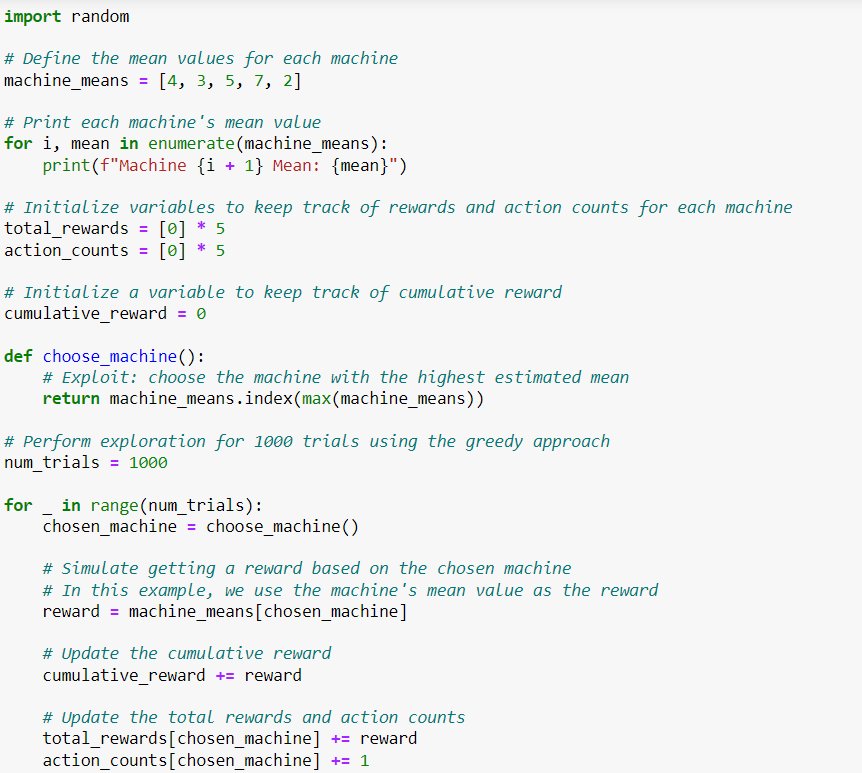
**Student Work Area**

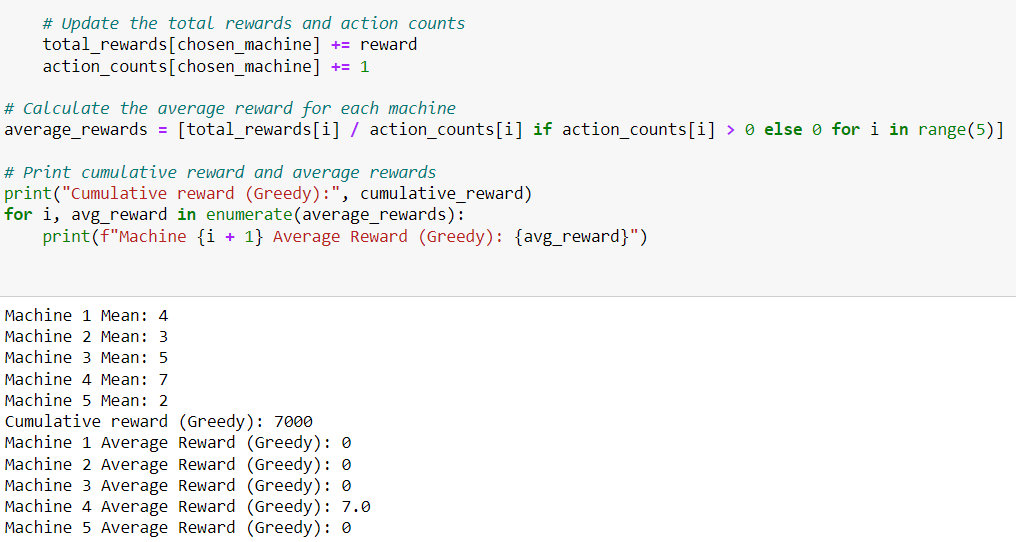
**Algorithm/Flowchart/Code/Sample Outputs**

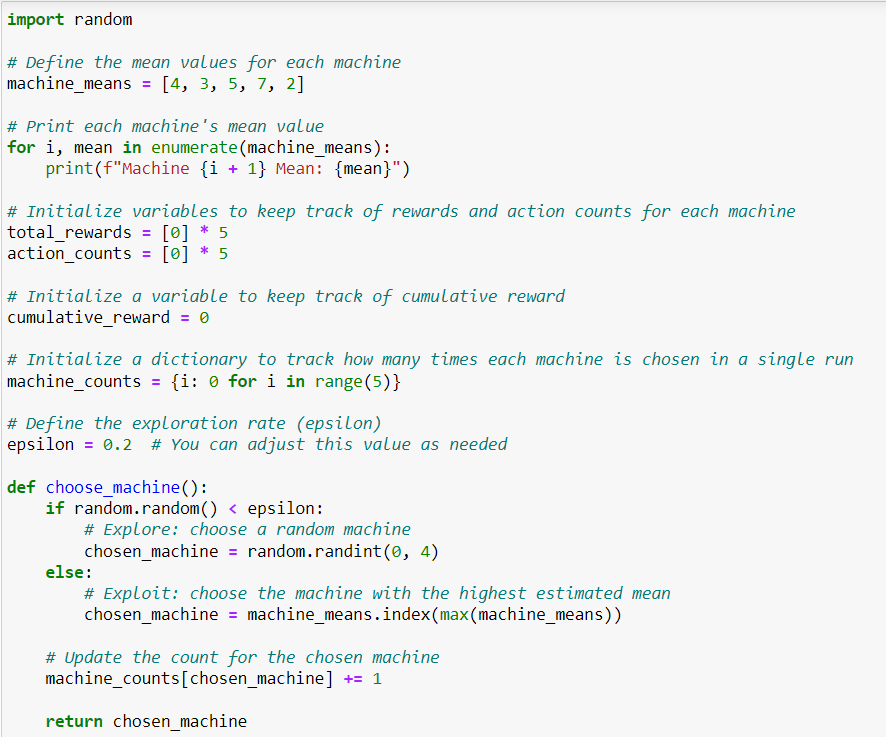
****

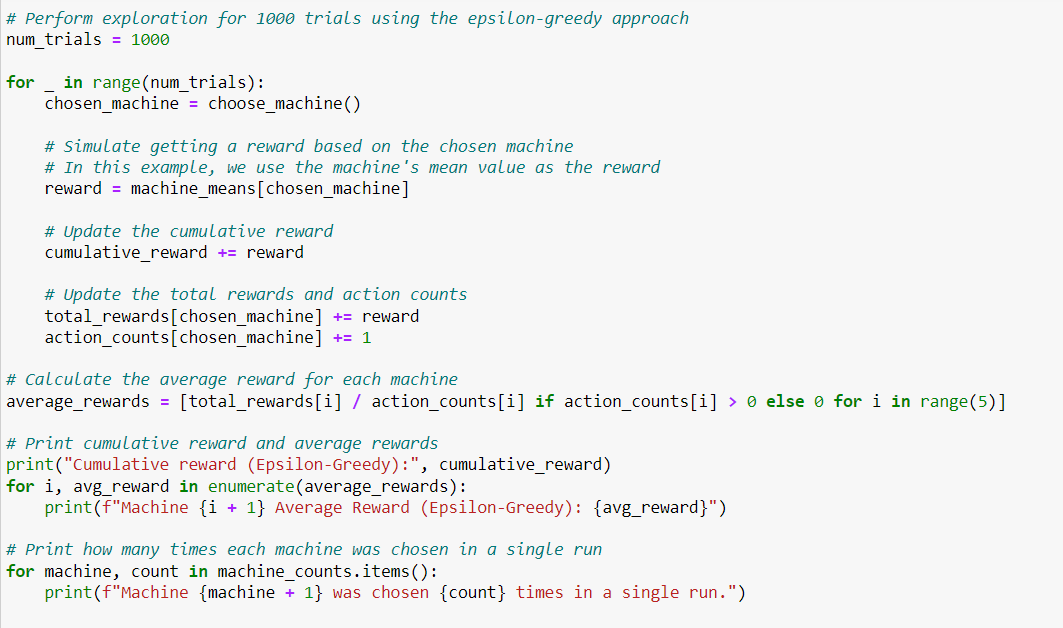
****

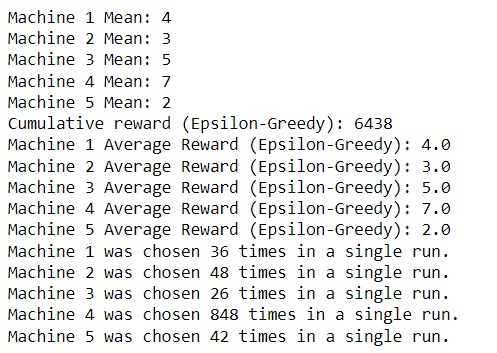
****

****

****

****

****

****

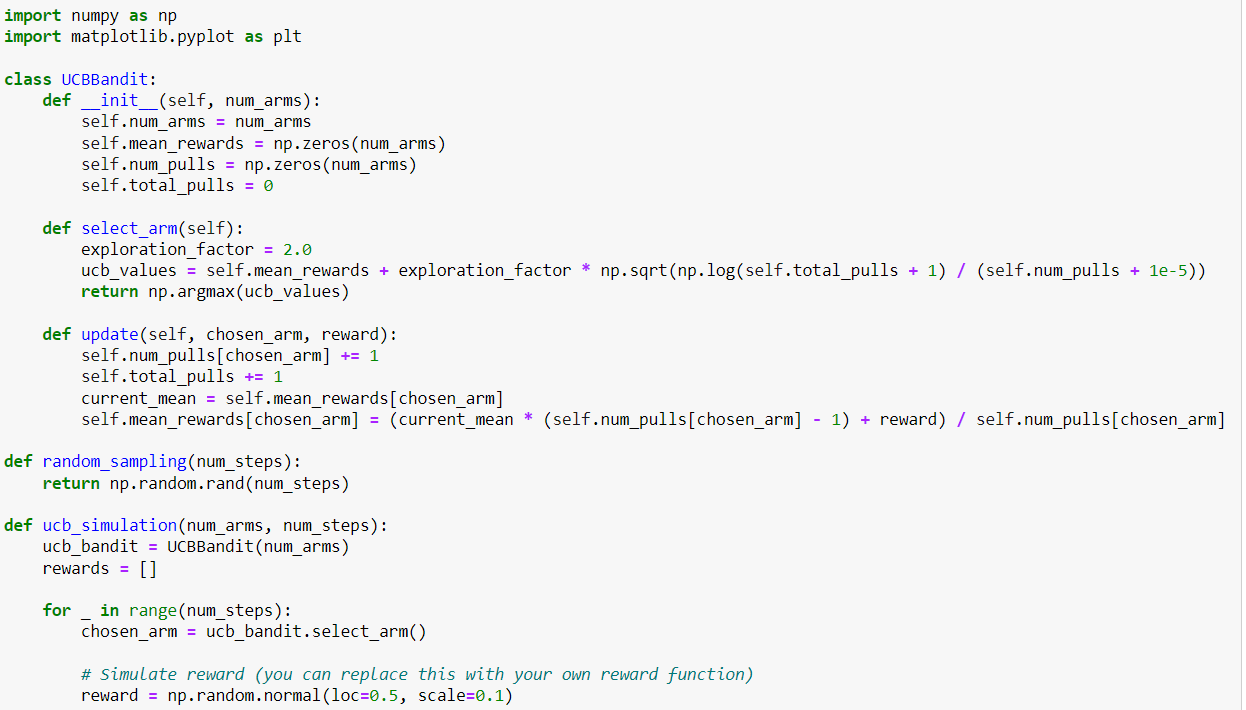
**EXPERIMENT NO. 4 and 5**

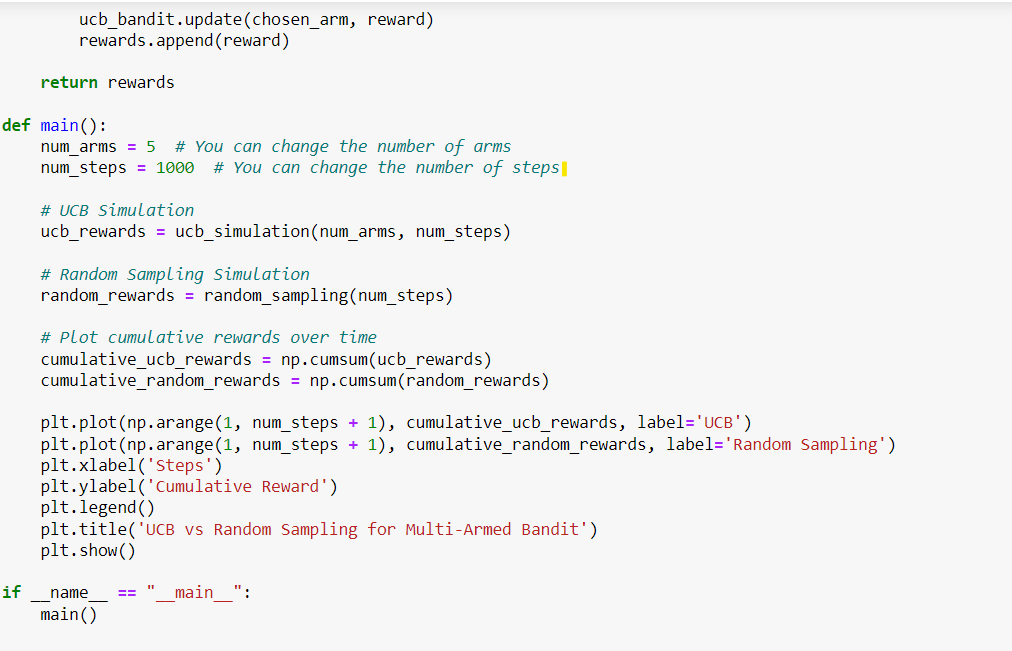
|  |
| --- |
| **Student Name and Roll Number: Aditya Sindhu 21csu278** |
| **Semester /Section:5th/AIML-B** |
| **Link to Code:** |
| **Date:20/9/23** |
| **Faculty Signature:** |
| **Marks/Grade:** |

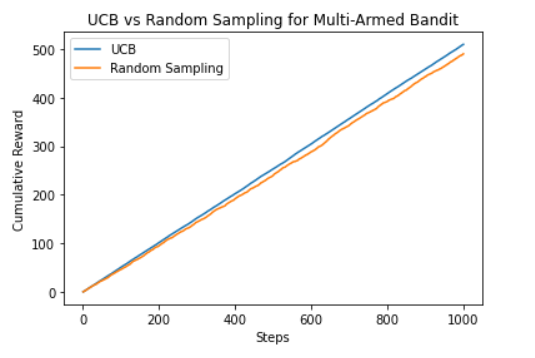
|  |
| --- |
| **Objective(s):** Solve the Multi-Armed Bandit Problem. |
| **Outcome:** Understand Thompson sampling as a solution to the Multi-Armed Bandit Problem. |
| **Problem Statement:** Write a python program to solve the Multi-Armed Bandit Problem using Thompson Sampling+ Write a python program to solve the multi armed bandit problem using the upper confidence bound algorithm. Compare the reward obtained with random 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 parametrized 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 modeled 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. Here's a comparison between these strategies:  1. \*\*Thompson Sampling\*\*:  - \*\*Probabilistic Approach\*\*: Thompson Sampling takes a probabilistic approach by modeling each arm's success probability as a probability distribution (often a beta distribution for binary rewards).  - \*\*Uncertainty Handling\*\*: It naturally handles uncertainty in the success probabilities. It updates beliefs based on observed outcomes and samples from these distributions for decision-making.  - \*\*Bayesian Framework\*\*: Thompson Sampling uses a Bayesian framework, where prior beliefs are updated with observed data using Bayes' theorem.  - \*\*Adaptive\*\*: It adapts quickly to changes in the environment or the underlying distribution of rewards.  - \*\*Exploration-Exploitation Trade-off\*\*: It maintains a balance between exploration and exploitation by randomly sampling from arm distributions.  - \*\*Suitable for Stochastic and Non-stochastic Environments\*\*: It works well in both stochastic and non-stochastic environments.  2. \*\*Epsilon-Greedy\*\*:  - \*\*Deterministic Approach\*\*: Epsilon-Greedy is a deterministic strategy. It chooses the arm with the highest estimated value with probability \(1 - \epsilon\) and explores with probability \(\epsilon\).  - \*\*No Uncertainty Handling\*\*: It does not inherently account for uncertainty in the rewards. It updates only the estimated mean of each arm.  - \*\*Simple to Implement\*\*: It is simple to implement and computationally efficient.  - \*\*Fixed Exploration Rate\*\*: The exploration rate \(\epsilon\) is typically fixed, which may not adapt well to changes in the environment.  - \*\*Less Flexible\*\*: It might not perform as well in complex or non-stationary environments.  3. \*\*UCB (Upper Confidence Bound)\*\*:  - \*\*Deterministic Approach\*\*: UCB also uses a deterministic approach. It chooses the arm with the highest upper confidence bound estimate.  - \*\*Handles Uncertainty\*\*: UCB addresses uncertainty by using confidence intervals to estimate the upper bound of the true mean of each arm.  - \*\*Balanced Exploration and Exploitation\*\*: It balances exploration and exploitation by favoring arms with higher uncertainty (exploration) and arms with high estimated values (exploitation).  - \*\*May Over-Exploit\*\*: In some cases, UCB can be overly optimistic and may over-exploit certain arms, especially in non-stationary environments.  - \*\*More Complex to Implement\*\*: It can be more complex to implement compared to Epsilon-Greedy.  In summary, each bandit strategy has its own strengths and weaknesses:  - \*\*Thompson Sampling\*\* is effective at handling uncertainty, adapts well to changes, and is suitable for both stochastic and non-stochastic environments.  - \*\*Epsilon-Greedy\*\* is simple to implement and computationally efficient, but it may not handle uncertainty as effectively as Thompson Sampling or UCB.  - \*\*UCB\*\* balances exploration and exploitation by considering upper confidence bounds, but it can be more complex to implement and may over-exploit in certain scenarios.  The choice of bandit strategy depends on the specific characteristics of the problem, the level of uncertainty in rewards, and computational considerations. Often, it's beneficial to experiment with different strategies to see which one performs best in a given context.   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.  By leveraging Bayesian principles, Thompson Sampling provides a principled and effective way to handle uncertainty in the multi-armed bandit setting, which is why it is often referred to as "Bayesian Bandits." |

**Student Work Area**

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

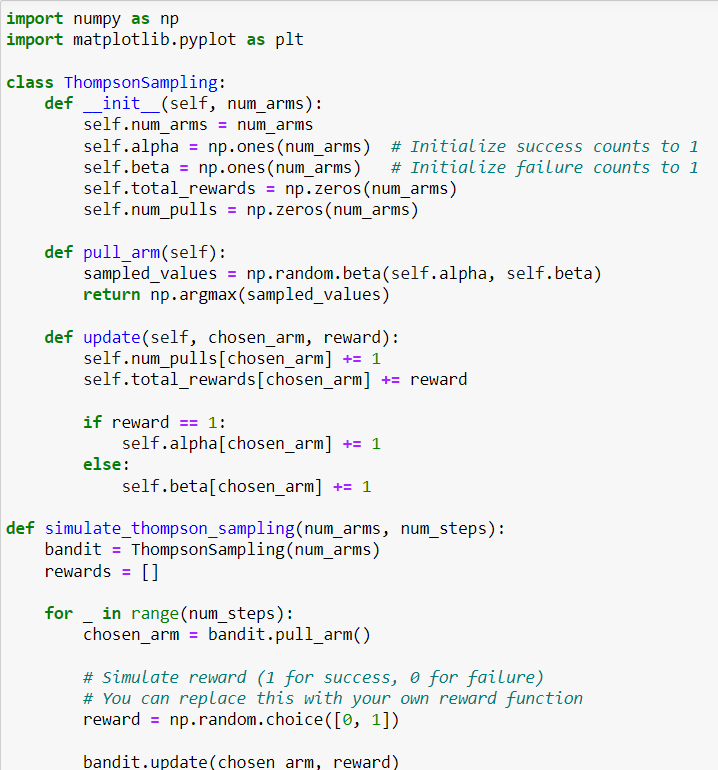
****

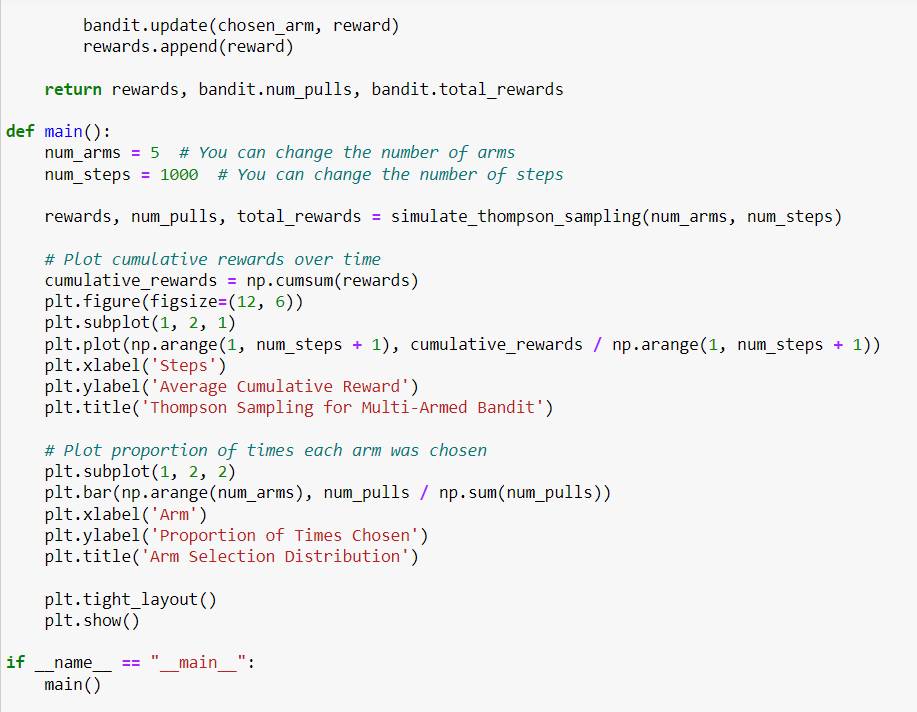
****

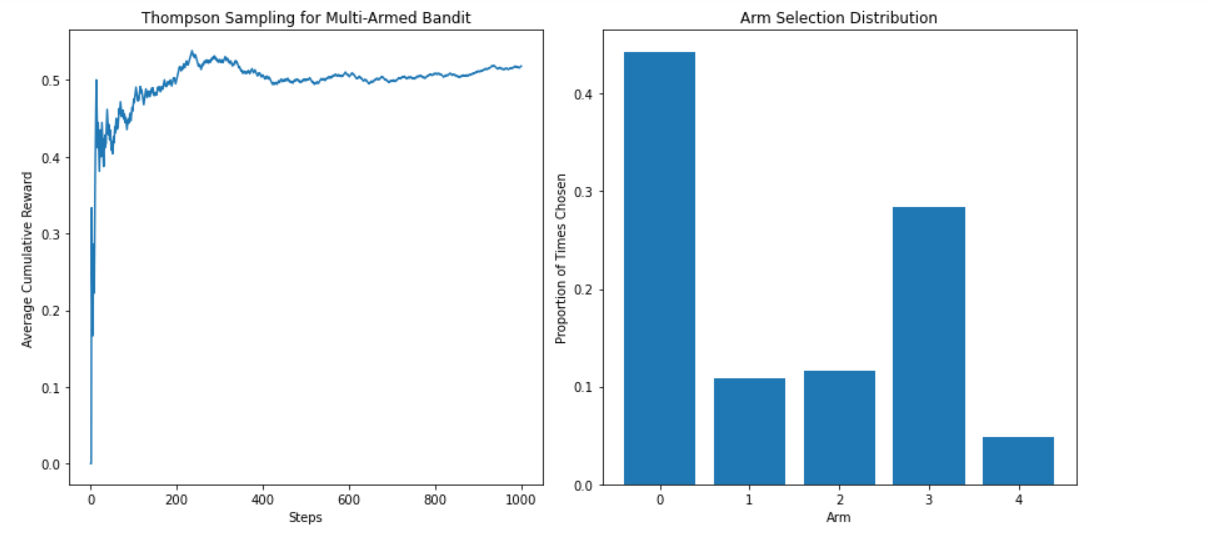
****

**===================================================================================================================================================================================================**

**==================================================================================================================================**

****

****

****

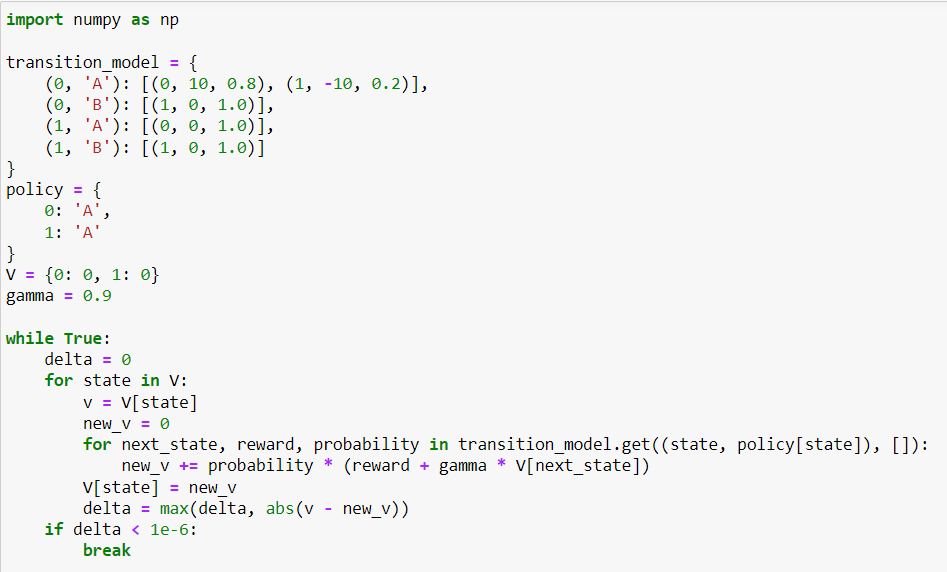
**EXPERIMENT NO. 6**

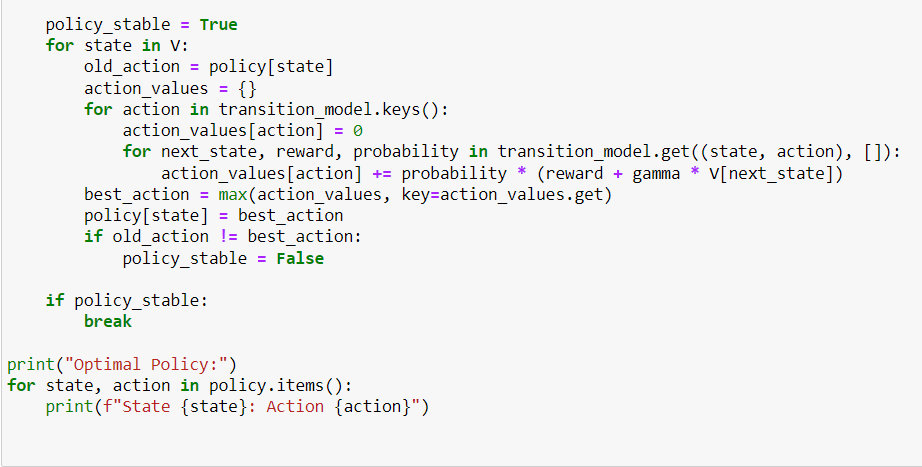
|  |
| --- |
| **Student Name and Roll Number: Aditya Sindhu 21csu278** |
| **Semester /Section:5th/AIML-B** |
| **Link to Code:** |
| **Date:**  **4/10/23** |
| **Faculty Signature:** |
| **Marks/Grade:** |

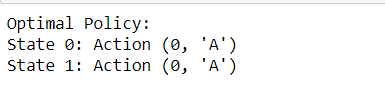
|  |
| --- |
| **Objective(s):** To understand the concept of dynamic programming and policy iteration in RL. |
| **Outcome:** Understand the policy iteration algorithm. |
| **Problem Statement:** Implementation of policy iteration algorithm in dynamic programming. |
| **Background Study:** Policy Iteration 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.  A policy is considered optimal under the following conditions:  1. \*\*Maximizes Expected Cumulative Reward\*\*: An optimal policy ensures that, when the agent follows it, the expected total reward obtained over time is greater than or equal to the expected total reward achievable with any other policy in the same environment.  2. \*\*Satisfies the Bellman Optimality Equation\*\*: An optimal policy satisfies the Bellman Optimality Equation, which describes the optimal value of states (or state-action pairs) in terms of the maximum expected return achievable under the policy. The Bellman Optimality Equation helps identify the values of states and actions that lead to the best outcomes.  For a state `s` or state-action pair `(s, a)`:    - Value of state `s` under the optimal policy (V\*(s)) is the maximum expected return.    - Value of taking action `a` in state `s` under the optimal policy (Q\*(s, a)) is the maximum expected return.    3. \*\*No Other Policy is Better\*\*: An optimal policy is superior to all other policies in terms of expected cumulative reward. There is no alternative policy that can consistently achieve a higher expected reward across all states.  In summary, an optimal policy is the most effective strategy an agent can adopt to achieve its goals in a given environment. It maximizes the expected long-term reward and satisfies the conditions set by the Bellman Optimality Equation, making it the best choice among all possible policies. Finding and implementing the optimal policy is a primary objective in reinforcement learning and decision-making tasks.   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**

****

****

****

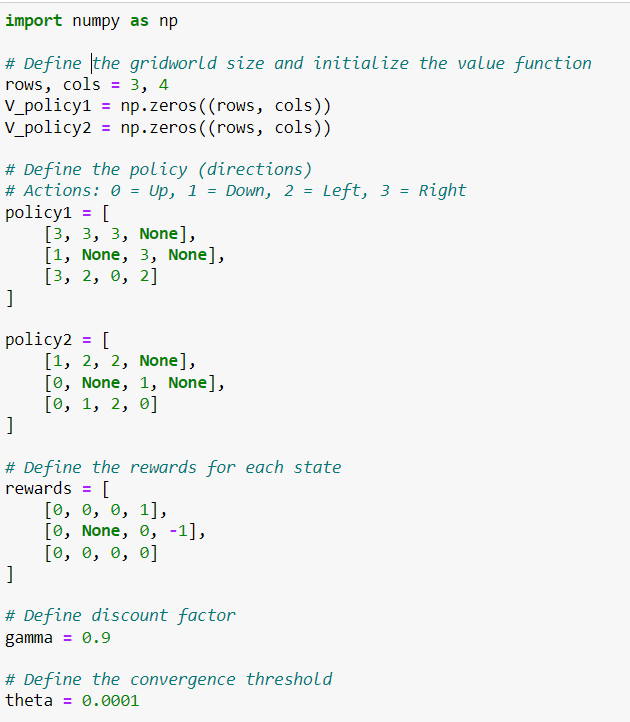
**EXPERIMENT NO. 7**

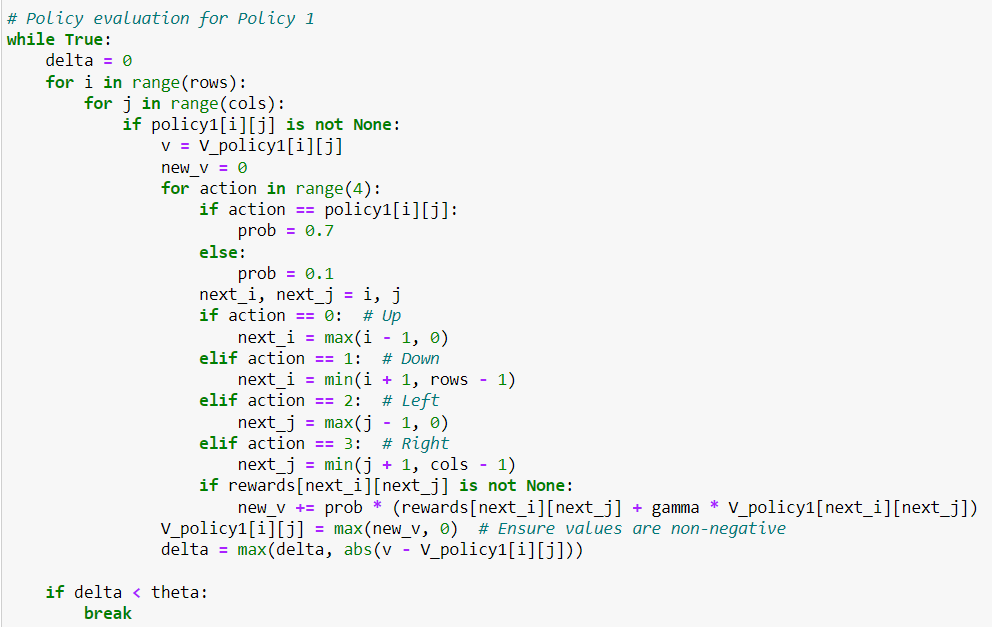
|  |
| --- |
| **Student Name and Roll Number: Keshav Bhatia** |
| **Semester /Section: 5th / AIML-B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

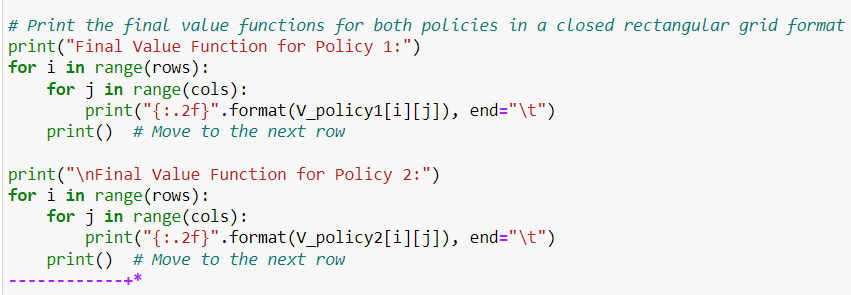
|  |
| --- |
| **Objective(s):** To understand the concepts of dynamic programming and value iteration in RL. |
| **Outcome:** Understand value iteration algorithm. |
| **Problem Statement:** Write a python program to implement value iteration in dynamic programming. |
| **Background Study:** One of 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. |

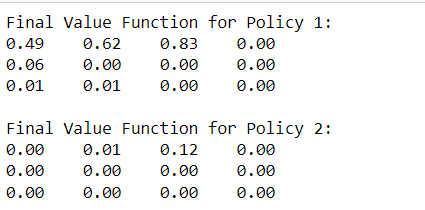
**Student Work Area**

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

****

****

****

****

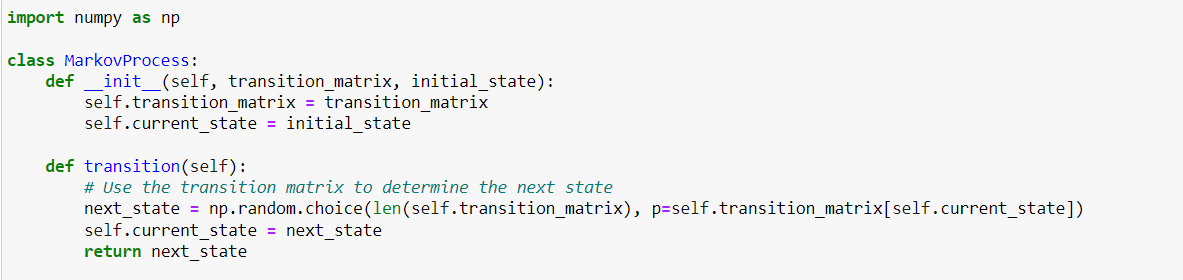
**EXPERIMENT NO. 8**

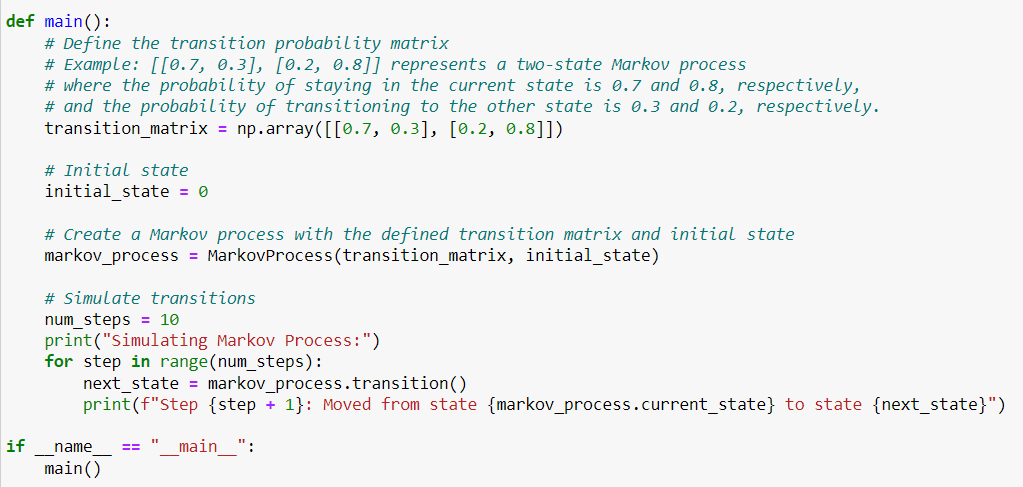
|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

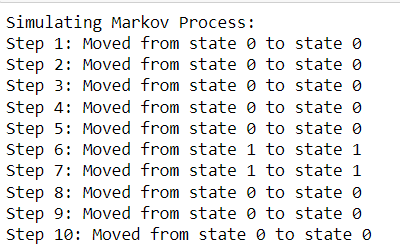
|  |
| --- |
| **Objective(s):** To understand Monte Carlo methods and apply them in Reinforcement Learning scenarios. |
| **Outcome:** Students will be familiarized with Monte Carlo methods. |
| **Problem Statement:** Write Python Program to implement Monte Carlo method to solve the Blackjack problem. |
| **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**

****

****

****

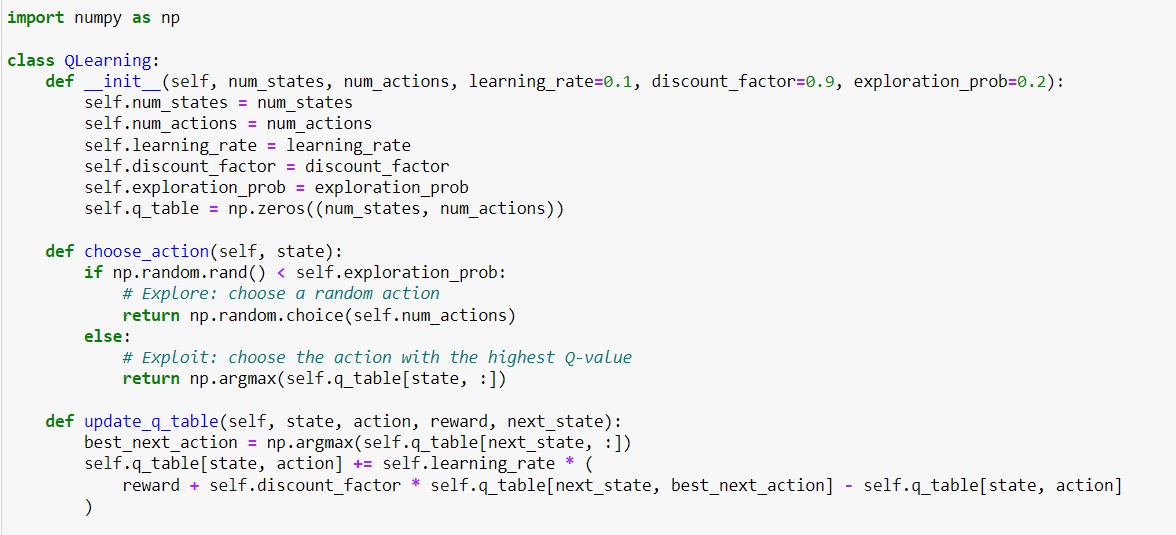
**EXPERIMENT NO. 9**

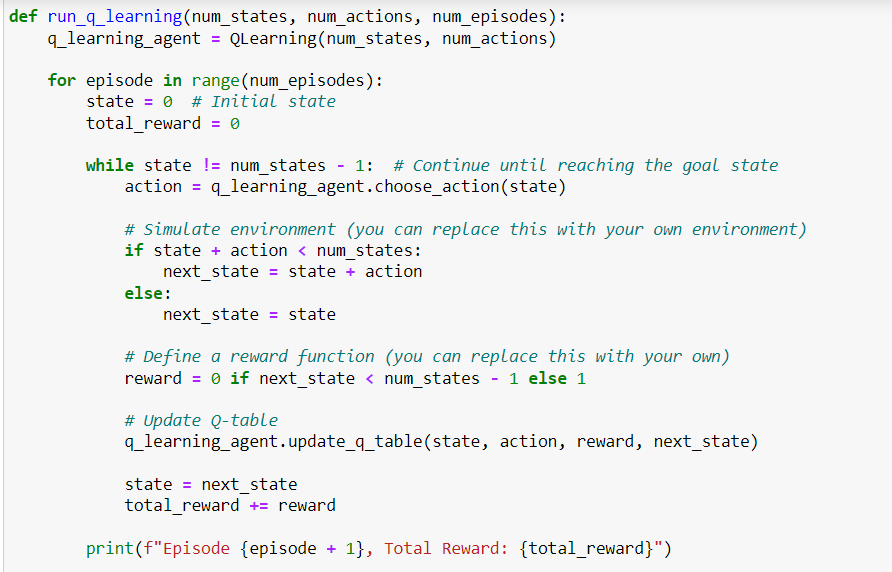
|  |
| --- |
| **Student Name and Roll Number: Aditya Sindhu 21csu278** |
| **Semester /Section: 5th / AIML-B** |
| **Link to Code:** |
| **Date:** |
| **Faculty Signature:** |
| **Marks/Grade:** |

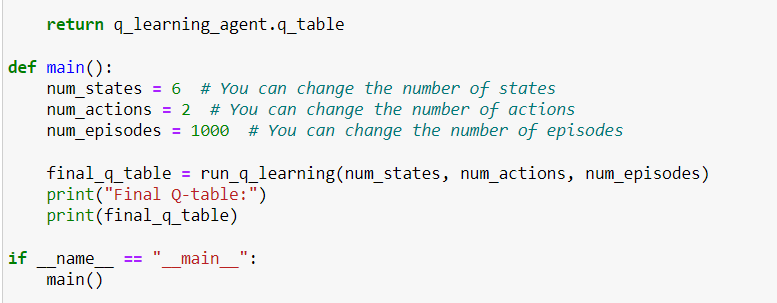
|  |
| --- |
| **Objective(s):** Write python program to implement Q-Learning |
| **Outcome(s):** To understand Q-Learning |
| **Problem Statement:** Implement Q-Learning using Python |
| **Background Study:** Q-learning is a model-free reinforcement learning algorithm to learn the value of an action 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? |

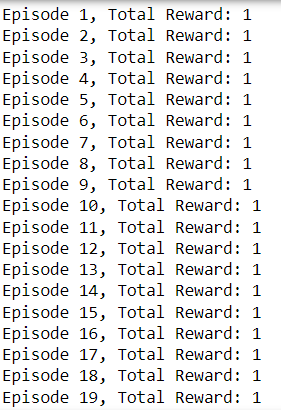
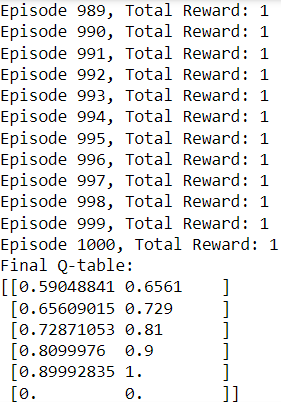
**Student Work Area**

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

****

****

****

**…………………………………..**

**Annexure 2**

**Reinforcement Learning**

**Project Report**



Faculty name Student name

Roll No.:

Semester:

Group:

Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122001, India

Session 2021-2022

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **S.No** |  | **Page No.** |
| **1.** | **Project Description** |  |
| **2.** | **Problem Statement** |  |
| **3.** | **Analysis**  **3.1 Hardware Requirements**  **3.2 Software Requirements** |  |
| **4.** | **Design**  **4.1 Data/Input Output Description:**  **4.2 Algorithmic Approach / Algorithm / DFD / ER diagram/Program Steps** |  |
| **5.** | **Implementation and Testing (stage/module wise)** |  |
| **6.** | **Output (Screenshots)** |  |
| **7.** | **Conclusion and Future Scope** |  |