Reinforcement Learning

Lab Manual

Department of Computer Science and Engineering
The NorthCap University, Gurugram

Reinforcement Learning
Laboratory Manual
CSL348

Dr. Monika Lamba

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

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

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.**

2. 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

3. GENERAL INSTRUCTIONS

a. 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.

b. 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**.**

c. 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

4. LIST OF EXPERIMENTS

Sr. No.	Title of the Experiment	Software used	Unit Covered	CO Covered	Time Required
1.	Implement Probability using Python	Python	1	CO1	2 Hours
2.	Write a Python Program to compute Karl Pearson and Spearman's Rank Correlation Coefficient	Python (Jupyter)	1	CO1	2 hours
3.	Write a Python program to solve the Multi-Armed Bandit problem using the Upper Confidence Bound Algorithm. Compare the reward obtained with random sampling.	Python (Jupyter)	2	CO2	2 hours
4.	Write a Python program to solve Multi-Armed Bandit problem using Thompson sampling.	Python (Jupyter)	2	CO2	2 hours
5.	Write a program to implement Q-Learning in Python.	Python (Jupyter)	3	CO3	2 hours
6.	Write python program to implement Markov Process.	Python (Jupyter)	3	CO3	2 hours
7.	Write a python program to implement policy iteration in Dynamic programming.	Python (Jupyter)	4	CO4	2 hours
8.	Write a python program to implement value iteration in Dynamic programming.	Python (Jupyter)	4	CO4	2 hours
9.	Write a Python Program to implement Monte Carlo method	Python (Jupyter)	5	CO5	2 hours
10	Write a Python Program to implement TD in Reinforcement Learning	Python (Jupyter)	5	CO5	2 hours
11	Implement function approximation methods	Python (Jupyter)	6	CO6	2 hours
12	Implement function approximation methods.	Python (Jupyter)	6	CO6	2 hours
Value Added Experiments					
13	Use RL algorithms to solve CartPole Balancing	Python (Jupyter)	5	CO2,3,4,5,6	2 hours
14	Create Deep Reinforcement Learning Algorithms to play Atari games.	Python (Jupyter)	5	CO2,3,4,5,6	2 hours

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

5. LIST OF FLIP EXPERIMENTS

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

6. LIST OF PROJECTS

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

7. 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: Neetu Singla

Student Name: Piyush Gambhir

Roll No.: 21CSU349

Semester: V

Group: AIML-B (AL-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: Piyush Gambir
Semester /Section: Semester V – AIML B
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL348-RL-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 08. 08.2023
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 Probability Distribution: This type of distribution deals with outcomes that are countable and distinct, often resulting from a finite or countably infinite set of possible values. Each value has a probability associated with it. Examples include the Binomial, Poisson, and Geometric distributions.

   Continuous Probability Distribution: Continuous distributions involve outcomes that can take on any value within a certain range. The probability of any single value is infinitesimally small, so probabilities are typically represented as areas under the probability density function (PDF) curve. Examples include the Normal (Gaussian), Exponential, and Uniform distributions.

2. Enlist some discrete probability distributions.
   - Binomial Distribution
   - Poisson Distribution
   - Geometric Distribution
   - Hypergeometric Distribution
   - Negative Binomial Distribution
   - Bernoulli Distribution

3. Enlist some continuous probability distributions.

- Normal (Gaussian) Distribution
- Exponential Distribution
- Uniform Distribution
- Gamma Distribution
- Beta Distribution
- Weibull Distribution

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Experiment 1

Problem Statement

Implement Probability Concepts on a Dataset.

Dataset Link

https://www.kaggle.com/datasets/uciml/student-alcohol-consumption

Code

In [ ]:
```
# import libraries
import pandas as pd
import scipy.stats as stats
from scipy.stats import uniform
from scipy.stats import poisson
```

In [ ]:
```
# reading the dataset
df = pd.read_csv('student-por.csv')

print(df.head().to_markdown())
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guardian	traveltime	studytime	failures	schoolsup	famsup	paid	activities	nursery	higher	internet	romantic	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	course	mother	2	2	0	yes	no	no	no	yes	yes	no	no	4	3	4	1	1	3	4	0	11	11
1	GP	F	17	U	GT3	T	1	1	at_home	other	course	father	1	2	0	no	yes	no	no	no	yes	yes	no	5	3	3	1	1	3	2	9	11	11
2	GP	F	15	U	LE3	T	1	1	at_home	other	other	mother	1	2	0	yes	no	no	no	yes	yes	yes	no	4	3	2	2	3	3	6	12	13	12
3	GP	F	15	U	GT3	T	4	2	health	services	home	mother	1	3	0	no	yes	no	yes	yes	yes	yes	yes	3	2	2	1	1	5	0	14	14	14
4	GP	F	16	U	GT3	T	3	3	other	other	home	father	1	2	0	no	yes	no	no	yes	yes	no	no	4	3	2	1	2	5	0	11	13	13

In [ ]:
```
# getting length of the dataset
print(f"Length of the dataset: {len(df)}")

# getting the number of columns
print(f"Number of columns: {len(df.columns)}")
```

```
# getting the number of rows
print(f"Number of rows: {len(df.index)}")
```

```
Length of the dataset: 649
Number of columns: 33
Number of rows: 649
```

In [ ]:
```
# Add a boolean column called grade_A noting if a student achieved 80% or higher as a final s
df['grade_A'] = df['G3'] * 5 >= 80
```

In [ ]:
```
# create a pivot table to see the relationship between absences and grade_A
pivot = df.iloc[0:395][['absences', 'G3']]
print(pivot.head().to_markdown())
```

	absences	G3
0	4	11
1	2	11
2	6	12
3	0	14
4	0	13

In [ ]:
```
# calculate count of students with absences > 10 and grade_A > 16
count = len(pivot[(pivot['absences'] > 10) & (pivot['G3'] > 16)])
print(f"Count of students with absences > 10 and grade_A > 16: {count}")

# calculate count of students with absences > 10
count1 = len(pivot[pivot['absences'] > 10])
print(f"Count of students with absences > 10: {count1}")

# calculate count of students with grade_A > 16
count2 = len(pivot[(pivot['G3'] > 16)])
print(f"Count of students with grade_A > 16: {count2}")
```

```
Count of students with absences > 10 and grade_A > 16: 1
Count of students with absences > 10: 37
Count of students with grade_A > 16: 26
```

In [ ]:
```
# calculate P(A ∩ B)
P_a_intersection_b = count/395
print(f"P(A ∩ B): {P_a_intersection_b}")

# calculate P(A)
P_a = count2/395
print(f"P(A): {P_a}")

# calculate P(B)
P_b = count1/395
print(f"P(B): {P_b}")
```

```
P(A ∩ B): 0.002531645569620253
P(A): 0.06582278481012659
P(B): 0.09367088607594937
```

In [ ]:
```
# calculate P(A|B)
P = P_a_intersection_b/P_b
print(f"P(A|B): {P}")
```

```
P(A|B): 0.02702702702702703
```

Question 1

THE BIRTHDAY PROBLEM

Compute the probability of getting a minimum of one overlapping birthday in a random group of 23 peoples. Because ontaining random sample again and again is tedious task, we can do simulations on a

computer with assumptions. The birthdays are independent of each other. Each possible birthday has the same probability. There are only 365 possible birthdays (not 366, as ignoring the leap year. Hint : in other words, we're modelling the process as drawing 23 independent samples from a discrete uniform distribution with parameter n = 365.

In [ ]:
```python
from random import randint


NUM_PEOPLE = 23
NUM_POSSIBLE_BIRTHDAYS = 365
NUM_TRIALS = 10000


def generate_random_birthday():
    birthday = randint(1, NUM_POSSIBLE_BIRTHDAYS)
    return birthday


def generate_k_birthdays(k):
    birthdays = [generate_random_birthday() for _ in range(k)]
    return birthdays


def aloc(birthdays):
    unique_birthdays = set(birthdays)

    num_birthdays = len(birthdays)
    num_unique_birthdays = len(unique_birthdays)
    has_coincidence = (num_birthdays != num_unique_birthdays)

    return has_coincidence


def estimate_p_aloc():
    num_aloc = 0
    for _ in range(NUM_TRIALS):
        birthdays = generate_k_birthdays(NUM_PEOPLE)
        has_coincidence = aloc(birthdays)
        if has_coincidence:
            num_aloc += 1

    p_aloc = num_aloc / NUM_TRIALS
    return p_aloc


p_aloc = estimate_p_aloc()
print(f"Estimated P(ALOC) after {NUM_TRIALS} trials: {p_aloc}")
```

```
Estimated P(ALOC) after 10000 trials: 0.5102
```

Question 2

The weight of certain species of frog is uniformly distributed from 15 and 25 grams. if you randomly select a frog, what is the probability that the frog weights between 17 and 19 grams.

In [ ]:
```python
# uniform distribution

p_uniform = uniform.cdf(x=19, loc=15, scale=10) - \
    uniform.cdf(x=17, loc=15, scale=10)

print(f"Probability of a uniform distribution: {p_uniform}")
```

```
Probability of a uniform distribution: 0.2
```

Question 3

Telecommunication Industry

According to the Telecommunication Industry the average monthly cell phone bill is Rs. 1000 with a standard deviation of Rs. 200.

What is the probability that a randomly selected cell phone bill is more than Rs 1200? What is the probability that a randomly selected cell phone bill is between Rs 750 and Rs 1200? What is the probability that a randomly selected cell phone bill is no more than Rs 650? What is the amount above which lies top 15% of cell phone bills? What is the amount below which lies bottom 25% of cell phone bills?

Note: This is a problem of normal probability distribution. Though the distribution is not mentioned, in absence of any other information we assume normality in the population.

In [ ]:
```
# normal distribution

a = stats.norm.cdf(1200, 1000, 200)
g = 1-a
print(g)

b = stats.norm.cdf(750, 1000, 200)
print(a-g)
stats.norm.ppf(.15, 1000, 200)

stats.norm.ppf(.25, 1000, 200)
```

```
0.15865525393145707
0.6826894921370859
```

Out[ ]:
```
865.1020499607837
```

Question 4

Fruit problem

Suppose we own a fruit shop and on an average 3 customers arrive in the shop every 10 minutes. The mean rate here is 3 or λ = 3. Poisson probability distributions can help us answer questions like what is the probability that 5 customers will arrive in the next 10 mins?

In [ ]:
```
# poisson distribution

# Probability of 5 or less customers
p = poisson.cdf(5, 3)
print("Probability of 5 or less customers: ", p)
```

```
Probability of 5 or less customers:  0.9160820579686966
```

EXPERIMENT NO. 2

Student Name and Roll Number: Piyush Gambir
Semester /Section: Semester V – AIML B
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL348-RL-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 22. 08.2023
Faculty Signature:
Marks/Grade:

Objective(s):
Compute correlation for two given series

Outcome: Understanding the meaning of correlation

Problem Statement: Write a Python Program to compute Correlation Coefficient in a dataset.

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.

   Correlation: Correlation refers to a statistical measure that describes the strength and direction of a relationship between two variables. When variables are correlated, changes in one variable tend to be associated with changes in the other, but this doesn't imply a cause-and-effect relationship.

   Causation: Causation implies a direct cause-and-effect relationship between two variables, where changes in one variable directly lead to changes in the other. Establishing causation requires careful experimentation and consideration of potential confounding variables.

2. How to compute Spearman's rank correlation coefficient for repeated ranks.

   Spearman's rank correlation coefficient can be computed for repeated ranks using the following formula:

   $$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2-1)}$$

   where:

   - ρ is the Spearman's rank correlation coefficient,
   - d_i is the difference between the paired ranks,
   - n is the number of paired ranks.

3. Elucidate on the graphical method for estimating correlation.

The graphical method involves plotting the paired data points on a scatter plot to visually assess the relationship between two variables.

Here's how it works:

- Plot each pair of data points as a point on the graph.
- If the points tend to form a pattern or trend (e.g., a straight line), there may be a correlation.
- The direction of the trend (upward or downward) suggests the correlation's direction (positive or negative).
- The closeness of the points to the trendline indicates the strength of the correlation.
- Be cautious: a strong correlation doesn't imply causation; other factors could be at play.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Experiment 2

Problem Statement

Implement Python Program to Compute Karl Pearson's and Spearman's Rank Correlation.

Dataset Link

https://www.kaggle.com/datasets/brendan45774/test-file

Code

In [ ]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Perfroming on Random Generated Data

In [ ]:
```
# setting the seed
np.random.seed(42)
```

In [ ]:
```
# generating a list x with 100 random numbers
x = np.random.rand(100)
print("x: ", x)
```

```
x:  [0.37454012 0.95071431 0.73199394 0.59865848 0.15601864 0.15599452
 0.05808361 0.86617615 0.60111501 0.70807258 0.02058449 0.96990985
 0.83244264 0.21233911 0.18182497 0.18340451 0.30424224 0.52475643
 0.43194502 0.29122914 0.61185289 0.13949386 0.29214465 0.36636184
 0.45606998 0.78517596 0.19967378 0.51423444 0.59241457 0.04645041
 0.60754485 0.17052412 0.06505159 0.94888554 0.96563203 0.80839735
 0.30461377 0.09767211 0.68423303 0.44015249 0.12203823 0.49517691
 0.03438852 0.9093204  0.25877998 0.66252228 0.31171108 0.52006802
 0.54671028 0.18485446 0.96958463 0.77513282 0.93949894 0.89482735
 0.59789998 0.92187424 0.0884925  0.19598286 0.04522729 0.32533033
 0.38867729 0.27134903 0.82873751 0.35675333 0.28093451 0.54269608
 0.14092422 0.80219698 0.07455064 0.98688694 0.77224477 0.19871568
 0.00552212 0.81546143 0.70685734 0.72900717 0.77127035 0.07404465
 0.35846573 0.11586906 0.86310343 0.62329813 0.33089802 0.06355835
 0.31098232 0.32518332 0.72960618 0.63755747 0.88721274 0.47221493
 0.11959425 0.71324479 0.76078505 0.5612772  0.77096718 0.4937956
 0.52273283 0.42754102 0.02541913 0.10789143]
```

In [ ]:
```
# generate a list y_positive with postive corelation with x
y_positive = x + np.random.rand(100)
print("y_positive: ", y_positive)
print("\n")

# generate a list y_negative with negative corelation with x
y_negative = 1 - x + np.random.rand(100)
print("y_negative: ", y_negative)
print("\n")

# generate a list y_no_corelation with no corelation with x
```

```
y_no_corelation = np.random.rand(100)
print("y_no_corelation: ", y_no_corelation)
print("\n")
```

```
y_positive:  [0.4059693  1.58712472 1.04634992 1.10722918 1.06358511 0.40528675
 0.46846654 1.62172728 0.82991318 0.78505249 0.31033595 1.13113114
 1.76214029 1.02045949 0.81522872 1.0548651  1.10791432 0.71132649
 1.32450402 0.83057138 1.41929305 1.03558516 0.61014812 0.47641377
 0.68400515 1.21228375 1.01768855 1.37496502 0.5993667  0.55719772
 1.02495586 0.39263193 0.18491696 1.28650071 1.90854174 1.13160028
 0.82340439 0.80069107 1.04786263 1.41193458 1.08448553 0.74695921
 0.53163703 1.21019871 0.54362048 0.69940923 0.92127541 1.02274704
 0.59818903 0.46350092 1.87785051 1.01469471 1.08439381 1.38428011
 1.58355043 1.16392951 0.76062805 0.95760248 0.28286483 1.05354668
 0.75646042 0.90365486 1.46226722 0.89252801 0.37122428 1.37799858
 0.46170429 0.98871549 0.11532579 1.57777988 1.44980913 0.21530351
 0.51761518 1.0419572  1.35203013 0.9033736  1.46220808 0.46078
 1.29519572 0.25339    1.20416978 0.73677165 1.25559164 0.9408977
 0.56892395 0.98516737 1.54682838 1.19275828 1.41686332 0.71406722
 0.21269701 1.61046055 1.66120311 1.19437865 1.10999697 0.84300517
 1.24868851 1.32465128 0.91250555 0.88776697]


y_negative:  [1.26749153 0.13342566 0.42963477 1.2998957  1.45041042 0.85320253
 1.04338793 0.79732562 0.40394657 0.45273547 1.5281493  0.72198535
 0.81951862 1.0119302  1.53035425 1.05384458 1.02115746 1.22173497
 1.21768788 1.55799427 1.04576    1.42881474 0.80153012 1.00135396
 0.80913238 0.45881368 1.77333677 0.87886329 1.29963199 1.58468821
 1.18726645 1.33211297 1.51185229 0.54363216 0.22961095 0.91405477
 0.97615859 0.92664385 0.96123927 0.73695819 1.81842035 1.45875167
 1.88047587 0.4608383  0.75667663 1.26579628 1.11647307 1.4465868
 1.4169097  1.668155   0.32486426 0.60996491 0.91163773 0.42209465
 0.57159277 0.63492703 1.84766227 1.50004693 1.52483388 0.77184616
 1.22632994 1.71870482 0.31134651 1.16157633 1.59643856 1.19807253
 1.55609152 0.9002871  1.28494051 0.30670491 1.03711639 1.61139771
 1.8615502  1.09777912 0.80448506 0.77250913 1.02702483 1.57591928
 1.34350115 1.67992361 1.02690192 0.71469703 1.04468493 1.03042359
 1.26729782 0.71075895 0.73599184 0.90508716 0.39932851 1.11861834
 0.910906   0.3241034  1.06181551 0.79891344 0.35609333 1.02844766
 1.24726072 0.78828001 1.59747135 0.97745604]


y_no_corelation:  [0.05168172 0.53135463 0.54063512 0.6374299  0.72609133 0.97585208
 0.51630035 0.32295647 0.79518619 0.27083225 0.43897142 0.07845638
 0.02535074 0.96264841 0.83598012 0.69597421 0.40895294 0.17329432
 0.15643704 0.2502429  0.54922666 0.71459592 0.66019738 0.2799339
 0.95486528 0.73789692 0.55435405 0.61172075 0.41960006 0.24773099
 0.35597268 0.75784611 0.01439349 0.11607264 0.04600264 0.0407288
 0.85546058 0.70365786 0.47417383 0.09783416 0.49161588 0.47347177
 0.17320187 0.43385165 0.39850473 0.6158501  0.63509365 0.04530401
 0.37461261 0.62585992 0.50313626 0.85648984 0.65869363 0.16293443
 0.07056875 0.64241928 0.02651131 0.58577558 0.94023024 0.57547418
 0.38816993 0.64328822 0.45825289 0.54561679 0.94146481 0.38610264
 0.96119056 0.90535064 0.19579113 0.0693613  0.100778   0.01822183
 0.09444296 0.68300677 0.07118865 0.31897563 0.84487531 0.02327194
 0.81446848 0.28185477 0.11816483 0.69673717 0.62894285 0.87747201
 0.73507104 0.80348093 0.28203457 0.17743954 0.75061475 0.80683474
 0.99050514 0.41261768 0.37201809 0.77641296 0.34080354 0.93075733
 0.85841275 0.42899403 0.75087107 0.75454287]
```

In [ ]:
```
# making a dataframe with x, y_positive, y_negative, y_no_corelation as columns
df = pd.DataFrame({"x": x, "y_positive": y_positive, "y_negative": y_negative, "y_no_corelati
print("df:\n\n", df.to_markdown())
```

df:

	x	y_positive	y_negative	y_no_corelation
0	0.37454	0.405969	1.26749	0.0516817
1	0.950714	1.58712	0.133426	0.531355
2	0.731994	1.04635	0.429635	0.540635
3	0.598658	1.10723	1.2999	0.63743
4	0.156019	1.06359	1.45041	0.726091
5	0.155995	0.405287	0.853203	0.975852
6	0.0580836	0.468467	1.04339	0.5163
7	0.866176	1.62173	0.797326	0.322956
8	0.601115	0.829913	0.403947	0.795186
9	0.708073	0.785052	0.452735	0.270832
10	0.0205845	0.310336	1.52815	0.438971
11	0.96991	1.13113	0.721985	0.0784564
12	0.832443	1.76214	0.819519	0.0253507
13	0.212339	1.02046	1.01193	0.962648
14	0.181825	0.815229	1.53035	0.83598
15	0.183405	1.05487	1.05384	0.695974
16	0.304242	1.10791	1.02116	0.408953
17	0.524756	0.711326	1.22173	0.173294
18	0.431945	1.3245	1.21769	0.156437
19	0.291229	0.830571	1.55799	0.250243
20	0.611853	1.41929	1.04576	0.549227
21	0.139494	1.03559	1.42881	0.714596
22	0.292145	0.610148	0.80153	0.660197
23	0.366362	0.476414	1.00135	0.279934
24	0.45607	0.684005	0.809132	0.954865
25	0.785176	1.21228	0.458814	0.737897
26	0.199674	1.01769	1.77334	0.554354
27	0.514234	1.37497	0.878863	0.611721
28	0.592415	0.599367	1.29963	0.4196
29	0.0464504	0.557198	1.58469	0.247731
30	0.607545	1.02496	1.18727	0.355973
31	0.170524	0.392632	1.33211	0.757846
32	0.0650516	0.184917	1.51185	0.0143935
33	0.948886	1.2865	0.543632	0.116073
34	0.965632	1.90854	0.229611	0.0460026
35	0.808397	1.1316	0.914055	0.0407288
36	0.304614	0.823404	0.976159	0.855461
37	0.0976721	0.800691	0.926644	0.703658
38	0.684233	1.04786	0.961239	0.474174
39	0.440152	1.41193	0.736958	0.0978342
40	0.122038	1.08449	1.81842	0.491616
41	0.495177	0.746959	1.45875	0.473472
42	0.0343885	0.531637	1.88048	0.173202
43	0.90932	1.2102	0.460838	0.433852
44	0.25878	0.54362	0.756677	0.398505
45	0.662522	0.699409	1.2658	0.61585
46	0.311711	0.921275	1.11647	0.635094
47	0.520068	1.02275	1.44659	0.045304
48	0.54671	0.598189	1.41691	0.374613
49	0.184854	0.463501	1.66815	0.62586
50	0.969585	1.87785	0.324864	0.503136
51	0.775133	1.01469	0.609965	0.85649
52	0.939499	1.08439	0.911638	0.658694
53	0.894827	1.38428	0.422095	0.162934
54	0.5979	1.58355	0.571593	0.0705687
55	0.921874	1.16393	0.634927	0.642419
56	0.0884925	0.760628	1.84766	0.0265113
57	0.195983	0.957602	1.50005	0.585776
58	0.0452273	0.282865	1.52483	0.94023
59	0.32533	1.05355	0.771846	0.575474
60	0.388677	0.75646	1.22633	0.38817
61	0.271349	0.903655	1.7187	0.643288

62	0.828738	1.46227	0.311347	0.458253
63	0.356753	0.892528	1.16158	0.545617
64	0.280935	0.371224	1.59644	0.941465
65	0.542696	1.378	1.19807	0.386103
66	0.140924	0.461704	1.55609	0.961191
67	0.802197	0.988715	0.900287	0.905351
68	0.0745506	0.115326	1.28494	0.195791
69	0.986887	1.57778	0.306705	0.0693613
70	0.772245	1.44981	1.03712	0.100778
71	0.198716	0.215304	1.6114	0.0182218
72	0.00552212	0.517615	1.86155	0.094443
73	0.815461	1.04196	1.09778	0.683007
74	0.706857	1.35203	0.804485	0.0711886
75	0.729007	0.903374	0.772509	0.318976
76	0.77127	1.46221	1.02702	0.844875
77	0.0740447	0.46078	1.57592	0.0232719
78	0.358466	1.2952	1.3435	0.814468
79	0.115869	0.25339	1.67992	0.281855
80	0.863103	1.20417	1.0269	0.118165
81	0.623298	0.736772	0.714697	0.696737
82	0.330898	1.25559	1.04468	0.628943
83	0.0635584	0.940898	1.03042	0.877472
84	0.310982	0.568924	1.2673	0.735071
85	0.325183	0.985167	0.710759	0.803481
86	0.729606	1.54683	0.735992	0.282035
87	0.637557	1.19276	0.905087	0.17744
88	0.887213	1.41686	0.399329	0.750615
89	0.472215	0.714067	1.11862	0.806835
90	0.119594	0.212697	0.910906	0.990505
91	0.713245	1.61046	0.324103	0.412618
92	0.760785	1.6612	1.06182	0.372018
93	0.561277	1.19438	0.798913	0.776413
94	0.770967	1.11	0.356093	0.340804
95	0.493796	0.843005	1.02845	0.930757
96	0.522733	1.24869	1.24726	0.858413
97	0.427541	1.32465	0.78828	0.428994
98	0.0254191	0.912506	1.59747	0.750871
99	0.107891	0.887767	0.977456	0.754543

In [ ]:
```
# computing corelation coefficient for x and y_positive, y_negative, y_no_corelation

# corelation coefficient for x and y_positive
corr_x_y_positive = np.corrcoef(x, y_positive)
print("corr_x_y_positive: ", corr_x_y_positive)
print("\n")

# corelation coefficient for x and y_negative
corr_x_y_negative = np.corrcoef(x, y_negative)
print("corr_x_y_negative: ", corr_x_y_negative)
print("\n")

# corelation coefficient for x and y_no_corelation
corr_x_y_no_corelation = np.corrcoef(x, y_no_corelation)
print("corr_x_y_no_corelation: ", corr_x_y_no_corelation)
print("\n")
```

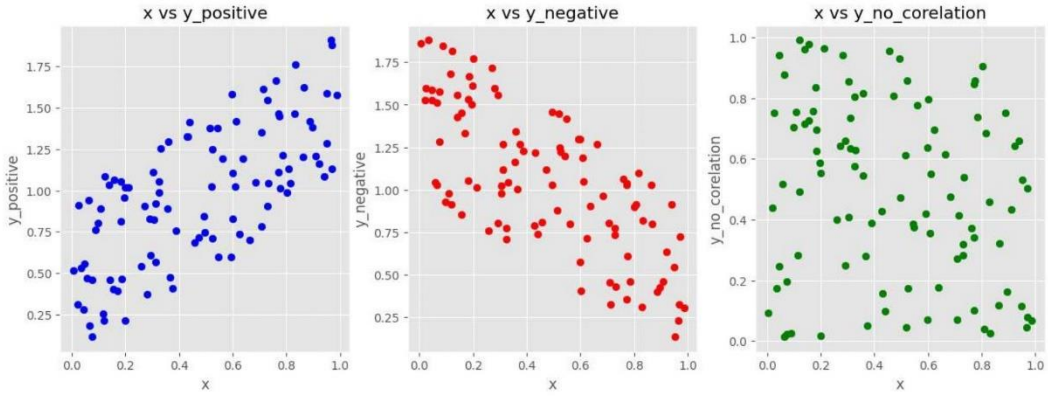
```
corr_x_y_positive:  [[1.         0.70046459]
 [0.70046459 1.        ]]


corr_x_y_negative:  [[ 1.         -0.72487394]
 [-0.72487394  1.        ]]


corr_x_y_no_corelation:  [[ 1.         -0.21188195]
 [-0.21188195  1.        ]]
```

In [ ]:
```
# correlation between x and y_positive, y_negative, y_no_corelation
print("correlation between x and y_positive: ", corr_x_y_positive[0, 1])
print("correlation between x and y_negative: ", corr_x_y_negative[0, 1])
print("correlation between x and y_no_corelation: ", corr_x_y_no_corelation[0, 1])
```

```
correlation between x and y_positive:  0.7004645930102837
correlation between x and y_negative:  -0.7248739439239781
correlation between x and y_no_corelation:  -0.2118819496524809
```

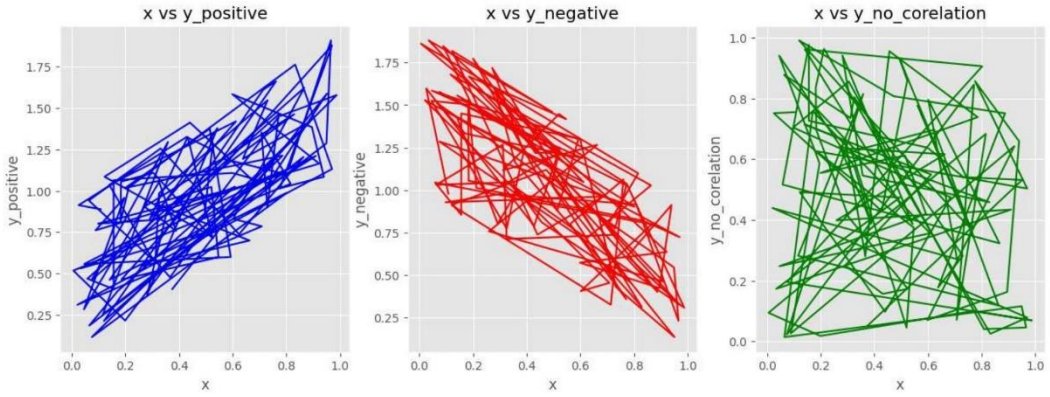
In [ ]:
```
# plotting x vs y_positive, y_negative, y_no_corelation using scatter plot
plt.figure(figsize=(15, 5))

# plotting x vs y_positive
plt.subplot(1, 3, 1)
plt.scatter(x, y_positive, color='blue')
plt.xlabel('x')
plt.ylabel('y_positive')
plt.title('x vs y_positive')

# plotting x vs y_negative
plt.subplot(1, 3, 2)
plt.scatter(x, y_negative, color='red')
plt.xlabel('x')
plt.ylabel('y_negative')
plt.title('x vs y_negative')


# plotting x vs y_no_corelation
plt.subplot(1, 3, 3)
plt.scatter(x, y_no_corelation, color='green')
plt.xlabel('x')
plt.ylabel('y_no_corelation')
plt.title('x vs y_no_corelation')
```

Out[ ]:
```
Text(0.5, 1.0, 'x vs y_no_corelation')
```



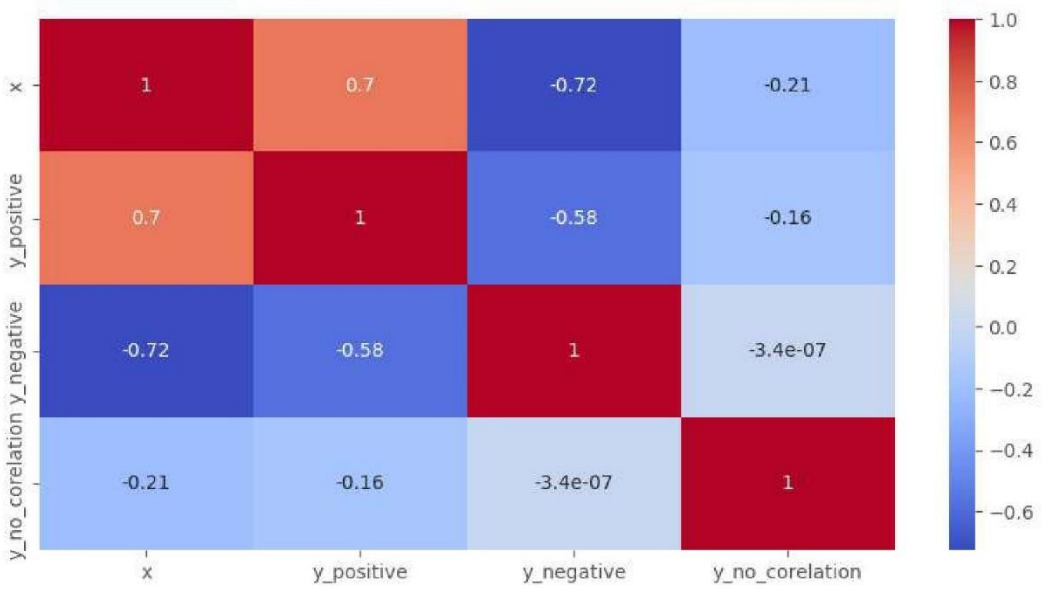
In [ ]:
```
# plotting x vs y_positive, y_negative, y_no_corelation using line plot
plt.figure(figsize=(15, 5))

# plotting x vs y_positive
plt.subplot(1, 3, 1)
plt.plot(x, y_positive, color='blue')
plt.xlabel('x')
plt.ylabel('y_positive')
plt.title('x vs y_positive')

# plotting x vs y_negative
plt.subplot(1, 3, 2)
plt.plot(x, y_negative, color='red')
plt.xlabel('x')
plt.ylabel('y_negative')
plt.title('x vs y_negative')

# plotting x vs y_no_corelation
plt.subplot(1, 3, 3)
plt.plot(x, y_no_corelation, color='green')
plt.xlabel('x')
plt.ylabel('y_no_corelation')
plt.title('x vs y_no_corelation')
```

Out[ ]:
```
Text(0.5, 1.0, 'x vs y_no_corelation')
```



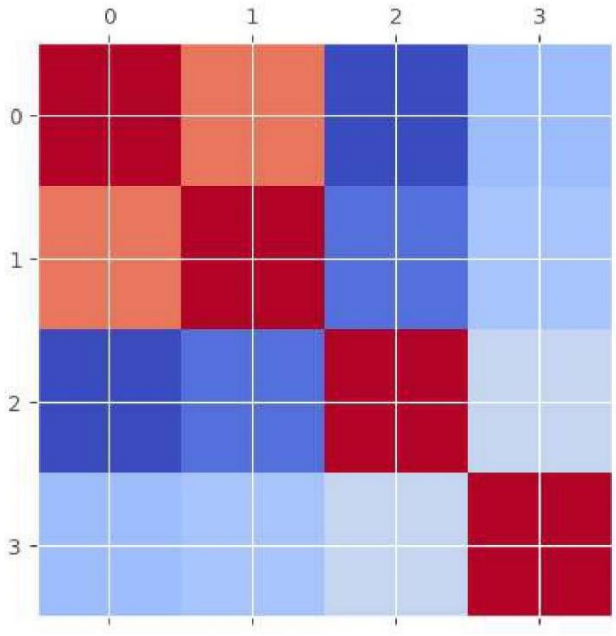
In [ ]:
```
# plotting heatmap for df
plt.figure(figsize=(10, 5))

# converting string to float
df['x'] = df['x'].astype(float)
df['y_positive'] = df['y_positive'].astype(float)
df['y_negative'] = df['y_negative'].astype(float)
df['y_no_corelation'] = df['y_no_corelation'].astype(float)

# converting df to correlation matrix
corr = df.corr()

# plotting heatmap
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[ ]:
```
<Axes: >
```



In [ ]:
```
# plotting matshow for df
plt.figure(figsize=(10, 5))

# converting df to correlation matrix
corr = df.corr()

# plotting matshow
plt.matshow(corr, cmap='coolwarm')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x11169303e50>
<Figure size 1000x500 with 0 Axes>
```

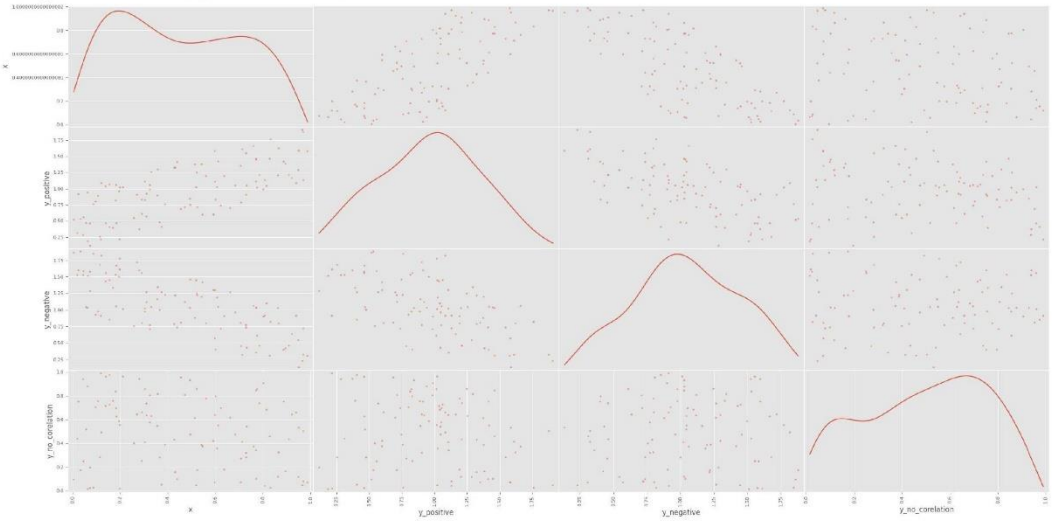


In [ ]:
```
# plotting scatter matrix for df
plt.figure(figsize=(30, 15))

# plotting scatter matrix
```

```
pd.plotting.scatter_matrix(df, figsize=(30, 15), diagonal='kde')

# plotting pairplot for df
plt.figure(figsize=(30, 15))
```

Out[ ]: <Figure size 3000x1500 with 0 Axes>

<Figure size 3000x1500 with 0 Axes>



<Figure size 3000x1500 with 0 Axes>

Performing on the dataset

In [ ]:
```
# reading the dataset
df_titanic = pd.read_csv('titanic.csv')

# printing the first 5 rows of the dataset
print("df_titanic:\n\n", df_titanic.head().to_markdown())
```

df_titanic:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	nan	Q
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47	1	0	363272	7	nan	S
2	894	0	2	Myles, Mr. Thomas Francis	male	62	0	0	240276	9.6875	nan	Q
3	895	0	3	Wirz, Mr. Albert	male	27	0	0	315154	8.6625	nan	S
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22	1	1	3101298	12.2875	nan	S

In [ ]:
```
# drop the unrequired columns
df_titanic.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
```

In [ ]:
```
# convert the sex columns to numerical data 0 or 1
df_titanic['Sex'] = df_titanic['Sex'].map({'male': 0, 'female': 1})
```

In [ ]:
```
# find qunique values in Embarked column
print("Unique values in Embarked column: ", df_titanic['Embarked'].unique())
```

```
# convert the Embarked column to numerical data 0, 1 or 2
df_titanic['Embarked'] = df_titanic['Embarked'].map({'S': 0, 'C': 1, 'Q': 2})
```

```
Unique values in Embarked column:  ['Q' 'S' 'C']
```

In [ ]:
```
# getting the updated dataset
print("df_titanic:\n\n", df_titanic.head().to_markdown())
```

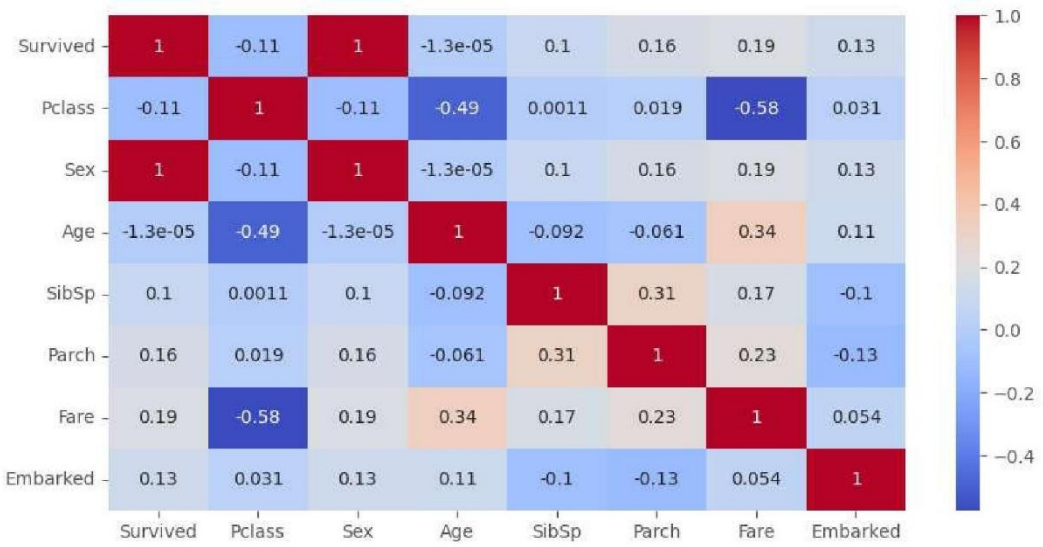
```
df_titanic:

|    |   Survived |   Pclass |   Sex |   Age |   SibSp |   Parch |    Fare |   Embarked |
|---:|-----------:|---------:|------:|------:|--------:|--------:|--------:|-----------:|
|  0 |          0 |        3 |     0 |  34.5 |       0 |       0 |  7.8292 |          2 |
|  1 |          1 |        3 |     1 |  47   |       1 |       0 |  7      |          0 |
|  2 |          0 |        2 |     0 |  62   |       0 |       0 |  9.6875 |          2 |
|  3 |          0 |        3 |     0 |  27   |       0 |       0 |  8.6625 |          0 |
|  4 |          1 |        3 |     1 |  22   |       1 |       1 | 12.2875 |          0 |
```

In [ ]:
```
# getting the correlation matrix
pearson_correlation = df_titanic.corr(method='pearson')
pearson_correlation
```

Out[ ]:

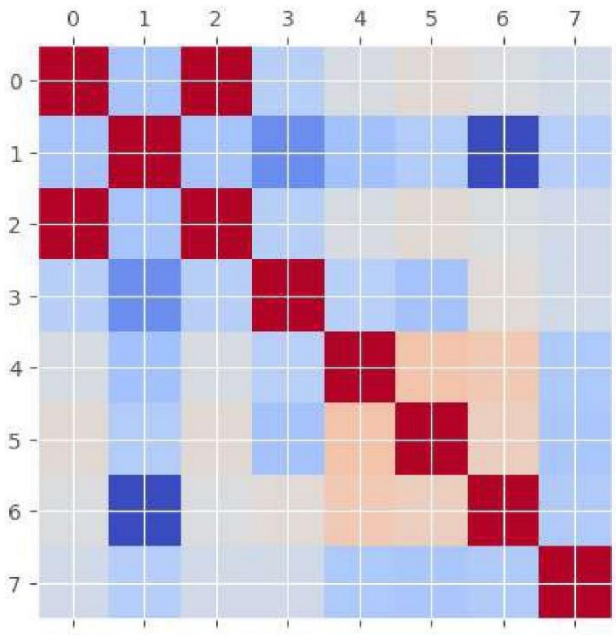
	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
Survived	1.000000	-0.108615	1.000000	-0.000013	0.099943	0.159120	0.191514	0.126779
Pclass	-0.108615	1.000000	-0.108615	-0.492143	0.001087	0.018721	-0.577147	0.031096
Sex	1.000000	-0.108615	1.000000	-0.000013	0.099943	0.159120	0.191514	0.126779
Age	-0.000013	-0.492143	-0.000013	1.000000	-0.091587	-0.061249	0.337932	0.113664
SibSp	0.099943	0.001087	0.099943	-0.091587	1.000000	0.306895	0.171539	-0.100603
Parch	0.159120	0.018721	0.159120	-0.061249	0.306895	1.000000	0.230046	-0.125164
Fare	0.191514	-0.577147	0.191514	0.337932	0.171539	0.230046	1.000000	0.053588
Embarked	0.126779	0.031096	0.126779	0.113664	-0.100603	-0.125164	0.053588	1.000000

In [ ]:
```
# plot the heatmap
plt.figure(figsize=(10, 5))

# plotting heatmap
sns.heatmap(pearson_correlation, annot=True, cmap='coolwarm')

# getting the correlation matrix
kendall_correlation = df_titanic.corr(method='kendall')
```



In [ ]:
```
# plot the correlation on matshow
plt.figure(figsize=(10, 5))

# plotting matshow
plt.matshow(kendall_correlation, cmap='coolwarm')

# getting the correlation matrix
spearman_correlation = df_titanic.corr(method='spearman')
```

```
<Figure size 1000x500 with 0 Axes>
```



EXPERIMENT NO. 3

Student Name and Roll Number: Piyush Gambir
Semester /Section: Semester V – AIML B
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL348-RL-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 05. 09.2023
Faculty Signature:
Marks/Grade:

Objective(s): Solve the Multi-Armed Bandit Problem
Outcome: Understanding and comparing bandit strategies.
Problem Statement: Write a Python program to solve the Multi-Armed Bandit problem using the Exploration only, Greedy Algorithm and Epsilon Greedy Algorithm.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Experiment 3

Problem Statement:

Write a Python program to solve the Multi-Armed Bandit problem using the Exploration only, Greedy Algorithm and Epsilon Greedy Algorithm.

Code

In [ ]:
```python
# importing required libraries
import numpy as np
```

In [ ]:
```python
# implementing the bandit class to create a bandit object
class Bandit:
    def __init__(self, name, estimated_reward):
        self.name = name
        self.estimated_reward = estimated_reward
        self.estimated_mean_reward = 0
        self.n = 0

    def pull(self):
        return np.random.randn() + self.estimated_reward
        # return np.random.randint(1, 10)

    def update(self, reward):
        self.n += 1
        self.estimated_mean_reward = (1 - 1.0/self.n) * \
            self.estimated_mean_reward + 1.0/self.n * reward
```

In [ ]:
```python
# implementing exploration by random sampling
def random_explore(bandits, num_iterations):
    for i in range(num_iterations):
        j = np.random.randint(0, len(bandits))
        x = bandits[j].pull()
        bandits[j].update(x)
```

In [ ]:
```python
# implementing exploration by mean sampling
def mean_explore(bandits, num_iterations):
    for i in range(num_iterations):
        j = np.argmax([b.estimated_mean_reward for b in bandits])
        x = bandits[j].pull()
        bandits[j].update(x)
```

In [ ]:
```python
# implementing the greedy_exploit function to return the best bandit
def greedy_exploit(bandits):
    return max(bandits, key=lambda x: x.estimated_mean_reward)

# implementing the epsilon greedy function to return the best bandit
def epsilon_greedy_exploit(bandits, epsilon=0.1):
    if np.random.random() < epsilon:
        # Randomly select a bandit with probability epsilon (exploration)
        return np.random.choice(bandits)
    else:
```

```
        # Exploit the bandit with the highest current estimated reward with probability 1-epsilo
        return max(bandits, key=lambda x: x.estimated_reward)
```

In [ ]:
```
# initialize machines A, B, C, D, E with true means between 5 and 10
np.random.seed(5)
machines = ['A', 'B', 'C', 'D', 'E']
# bandits = [Bandit(machine, 5 + 5 * np.random.rand()) for machine in machines]
estimated_rewards = [4, 3, 5, 7, 2]
bandits = [Bandit(machine, mean_reward) for machine, mean_reward in zip(machines, estimated_rewa
```

In [ ]:
```
# random_explore(bandits, 1000)
# print("Bandit Details Using Random\n")
# for bandit in bandits:
#     print("Bandit:", bandit.name, "Number of Times: ", bandit.n,"Estimated Mean Reward: ",
#           bandit.estimated_mean_reward)
```

In [ ]:
```
# initial pull for each bandit
all_rewards = []
for bandit in bandits:
    reward = bandit.pull()
    bandit.update(reward)
    all_rewards.append(reward)

# Number of trials
N = 1000 - len(bandits)
for _ in range(N):
    chosen_bandit = epsilon_greedy_exploit(bandits)
    reward = chosen_bandit.pull()
    chosen_bandit.update(reward)
    all_rewards.append(reward)

# calculating mean and cumulative reward
mean_rewards = [np.mean(all_rewards[:i+1]) for i in range(len(all_rewards))]
cumulative_rewards = np.cumsum(all_rewards).tolist()

print("Bandit Details Using Greedy Algorithm \n")
for bandit in bandits:
    print(
        f"Machine {bandit.name} - Number of Times {bandit.n} Estimated Reward: {bandit.estimated_

print(f"\nFinal Mean Reward: {mean_rewards[-1]:.2f}")
print(f"Final Cumulative Reward: {cumulative_rewards[-1]:.2f}")
```

```
Bandit Details Using Greedy Algorithm

Machine A - Number of Times 21 Estimated Reward: 4.00, Estimated Mean Reward: 4.42
Machine B - Number of Times 30 Estimated Reward: 3.00, Estimated Mean Reward: 2.84
Machine C - Number of Times 17 Estimated Reward: 5.00, Estimated Mean Reward: 5.33
Machine D - Number of Times 914 Estimated Reward: 7.00, Estimated Mean Reward: 7.03
Machine E - Number of Times 18 Estimated Reward: 2.00, Estimated Mean Reward: 2.24

Final Mean Reward: 6.73
Final Cumulative Reward: 6734.47
```

EXPERIMENT NO. 4

Student Name and Roll Number: Piyush Gambir
Semester /Section: Semester V – AIML B
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL348-RL-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 05. 09.2023
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: Involves trying out different options to gather information about their outcomes. It helps in discovering the characteristics of different choices and is crucial in uncertain environments.
   - Exploitation: Refers to choosing the option that is believed to be the best based on current knowledge. Exploitation maximizes short-term gain by selecting the seemingly optimal choice.

2. Differentiate between greedy and epsilon greedy strategies for solving Multi-armed bandit problem.
   - Greedy Strategy: Always selects the option with the highest estimated reward. It exploits current knowledge without exploring other options. This can lead to suboptimal choices if the initial estimates are inaccurate.
   - Epsilon Greedy Strategy: Balances exploration and exploitation by selecting the best-known option with probability $1-\epsilon$ and exploring a random option with probability ϵ. This ensures a balance between exploiting the best-known choices and exploring new options to refine estimates.

3. Explain the Upper Confidence Bound Algorithm for solving Multi-armed bandit problem.
    - Idea: UCB balances exploration and exploitation by choosing arms based on upper confidence bounds on their estimated rewards.
    - Selection Rule: At each step, the algorithm selects the arm with the highest upper confidence bound, which incorporates both the estimated mean reward and a term related to uncertainty or confidence in the estimate.
    - Advantage: UCB tends to prioritize arms that are not only estimated to have high rewards but also have higher uncertainty, encouraging exploration. As more data is collected, the uncertainty decreases, and the algorithm converges towards exploiting the best-performing arm.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Experiment 4

Problem Statement:

Implement a python program to solve the muti-armed bandit problem using the Upper Confidence Bound Algorithm. Compare the reward obtained with random sampling.

Code

In [ ]:
```python
# importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

In [ ]:
```python
# ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```python
# define the number of slot machines
num_machines = 5

# initialize the total number of trials
total_trials = 1000

# initialize the mean values for each machine
mean_values = np.random.randint(1, 10, size=num_machines)

# initialize the number of times each machine is pulled
num_pulls = np.zeros(num_machines)

# initialize the sum of rewards for each machine
sum_rewards = np.zeros(num_machines)

# initialize the total cumulative reward
total_cumulative_reward = 0

# UCB algorithm
for t in range(1, total_trials + 1):
    # calculate UCB values
    ucb_values = sum_rewards / num_pulls + np.sqrt(2 * np.log(t) / num_pulls)

    # determine machine with max UCB value
    machine_to_pull = np.argmax(ucb_values)

    # generate reward
    reward = np.random.normal(mean_values[machine_to_pull], 1)

    # update counters
    num_pulls[machine_to_pull] += 1
    sum_rewards[machine_to_pull] += reward

    # update total reward
    total_cumulative_reward += reward

print(f"UCB Total Cumulative Reward: {total_cumulative_reward}")
```

```
UCB Total Cumulative Reward: 8957.614307351323
```

EXPERIMENT NO. 5

Student Name and Roll Number: Piyush Gambir
Semester /Section: Semester V – AIML B
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL348-RL-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 12. 09.2023
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 Distribution:** A continuous probability distribution defined on the interval [0, 1]. It is characterized by two shape parameters, often denoted as α and β.
   - **Usage in Thompson Sampling:** Beta distributions are employed in Thompson sampling as priors to model uncertainty about the true probabilities of success for different arms in a Multi-armed Bandit problem. The distribution is updated based on observed outcomes, providing a Bayesian framework for decision-making.

2. Compare and contrast Thompson sampling with other bandit strategies.

   Thompson Sampling vs. Epsilon-Greedy:
   - Exploration-Exploitation Tradeoff: Thompson Sampling naturally balances exploration and exploitation, adapting its choices based on uncertainty. Epsilon-Greedy has a fixed exploration-exploitation tradeoff.
   - Learning Model: Thompson Sampling updates its belief about the arms' distributions through Bayesian inference, while Epsilon-Greedy relies on simple averages.

   Thompson Sampling vs. UCB (Upper Confidence Bound):
   - Uncertainty Handling: UCB uses deterministic upper confidence bounds, while Thompson Sampling incorporates probabilistic uncertainty through Bayesian updating.

- Theoretical Consideration: Thompson Sampling has favorable theoretical guarantees, especially in stochastic environments, but UCB is known for its simplicity and effectiveness.

3. Why is Thompson sampling referred to as Bayesian Bandits?
   - Bayesian Nature: Thompson Sampling is considered Bayesian because it employs Bayesian inference. It maintains a probability distribution (typically Beta distribution) over the unknown parameters (probability of success for each arm).
   - Updating Beliefs: After observing outcomes, Thompson Sampling updates its probability distribution based on Bayes' theorem, allowing the algorithm to incorporate prior knowledge and adapt its beliefs over time.
   - Probabilistic Decision-Making: Instead of deterministically selecting the arm with the highest estimated reward, Thompson Sampling probabilistically samples from the current distribution, making decisions that account for uncertainty in the underlying model.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Problem Statement:

Write a python program to solve the Multi-Armed Bandit Problem using Thompson Sampling.

Code

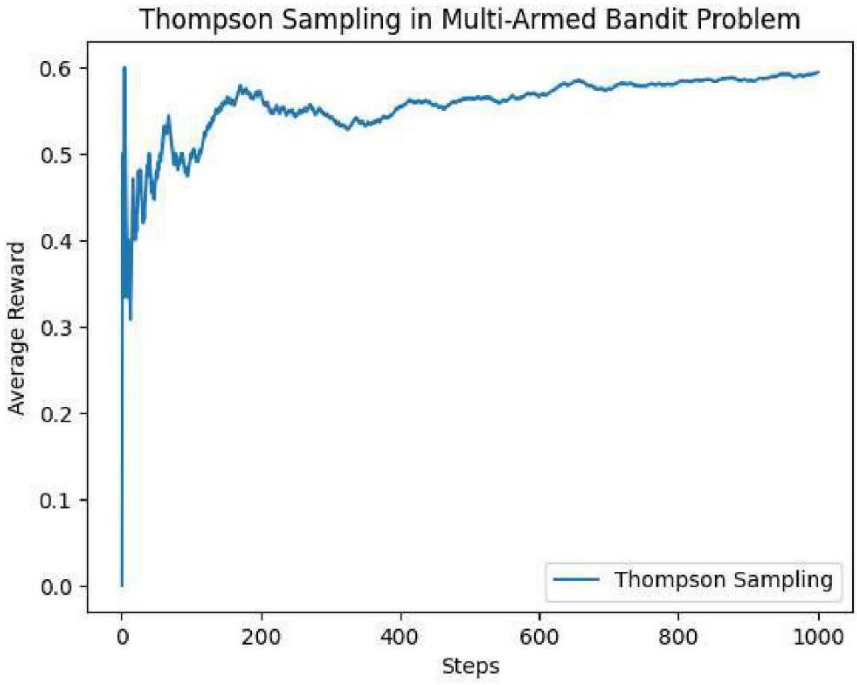
In [ ]:
```python
# importing required libraries
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```python
def thompson_sampling_bandit(arms, num_steps):
    num_arms = len(arms)
    successes = np.zeros(num_arms)
    failures = np.zeros(num_arms)

    total_reward_ts = 0
    rewards_ts = []

    for t in range(1, num_steps + 1):
        sampled_theta = np.random.beta(successes + 1, failures + 1)
        action = np.argmax(sampled_theta)

        reward = arms[action]()
        total_reward_ts += reward

        if reward == 1:
            successes[action] += 1
        else:
            failures[action] += 1

        rewards_ts.append(total_reward_ts / t)

    return rewards_ts
```

In [ ]:
```python
def generate_bandit_arms(num_arms, true_probs):
    return [lambda p=true_probs[i]: np.random.binomial(1, p) for i in range(num_arms)]
```

In [ ]:
```python
def plot_results(steps, ts_rewards):
    plt.plot(range(1, steps + 1), ts_rewards, label='Thompson Sampling')
    plt.xlabel('Steps')
    plt.ylabel('Average Reward')
    plt.legend()
    plt.title('Thompson Sampling in Multi-Armed Bandit Problem')
    plt.show()
```

In [ ]:
```python
if __name__ == '__main__':
    num_arms = 5
    true_probs = np.random.uniform(0.2, 0.8, num_arms)
    num_steps = 1000

    arms = generate_bandit_arms(num_arms, true_probs)
    ts_rewards = thompson_sampling_bandit(arms, num_steps)

    # Plot results
    plot_results(num_steps, ts_rewards)
```


Thompson Sampling in Multi-Armed Bandit Problem
Average Reward
Steps
Thompson Sampling
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
200
400
600
800
1000

EXPERIMENT NO. 6

Student Name and Roll Number: Piyush Gambir
Semester /Section: Semester V – AIML B
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL348-RL-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 19. 09.2023
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 equation is a framework for determining the optimal expected reward at a state. The Bellman optimality equation is the same as the Bellman expectation equation, but instead of taking the average of the actions, it takes the action with the maximum value.

2. What is dynamic programming?
   Dynamic programming is a method for solving complex problems by breaking them down into simpler overlapping subproblems. In the context of reinforcement learning, it involves solving subproblems related to the optimal policy or value function.

3. What is a policy?
   A policy in reinforcement learning is a mapping from states to actions. It defines the agent's strategy for decision-making in the environment.

4. Explain the policy evaluation and policy improvement steps in policy iteration.
   - **Policy Evaluation:** Given a policy, iteratively calculate the value function for each state until convergence. This involves applying the Bellman expectation equation repeatedly.
   - **Policy Improvement:** Based on the updated value function, greedily update the policy by choosing actions that maximize expected future rewards. This step is guided by the Bellman optimality equation.

5. What do you mean by optimal policy? When is a policy optimal?
   An optimal policy is a policy that, when followed, maximizes the expected cumulative reward over time. In other words, it leads to the highest possible value function.

6. What is the convergence condition for the policy iteration algorithm?
   Policy iteration converges when both the policy evaluation and policy improvement steps reach a stable state, indicating that the policy and value function have become optimal. The process stops when the policy no longer changes during the policy improvement step.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Problem Statement:

Implement Polciy Iteration Algorithm.

Code

In [ ]:
```
# importing required libraries
import numpy as np
```

In [ ]:
```
num_states = 11
num_actions = 4

states = [(i, j) for i in range(3) for j in range(4) if not (i == 1 and j == 1)]

policy = {
    (0, 0): 'Right',
    (0, 1): 'Right',
    (0, 2): 'Right',
    (1, 0): 'Up',
    (1, 2): 'Up',
    (2, 0): 'Up',
    (2, 1): 'Right',
    (2, 2): 'Up',
    (2, 3): 'Left',
}

rewards = {
    (0, 3): 1,
    (1, 3): -1,
}

def transition_probabilities(state, action):
    i, j = state
    if state in rewards:
        return [(state, 1.0)]
    if action == 'Up':
        next_state = (max(i - 1, 0), j)
    elif action == 'Down':
        next_state = (min(i + 1, 2), j)
    elif action == 'Left':
        next_state = (i, max(j - 1, 0))
    elif action == 'Right':
        next_state = (i, min(j + 1, 3))
    else:
        raise ValueError("Invalid action")
    if next_state not in states:
        return [(state, 1.0)]
    return [(next_state, 1.0)]

gamma = 0.9

def policy_evaluation(policy, states, rewards, gamma, theta):
    V = {state: 0 for state in states}
    iteration = 0
    while True:
        delta = 0
```

```
        for state in states:
            v = V[state]
            action = policy.get(state)
            if action:
                transitions = transition_probabilities(state, action)
                V[state] = sum(prob * (rewards.get(next_state, 0) + gamma * V.get(next_state,
            delta = max(delta, abs(v - V[state]))
        if delta < theta:
            break
        print(f"Iteration {iteration} - Value Matrix:")
        value_matrix = np.zeros((3, 4))
        for state, value in V.items():
            i, j = state
            value_matrix[i][j] = value
        print(value_matrix)
        iteration += 1
    return V

def policy_improvement(policy, V, states, rewards, gamma):
    policy_stable = True
    for state in states:
        old_action = policy.get(state)
        if old_action:
            action_values = {}
            for action in ['Up', 'Down', 'Left', 'Right']:
                transitions = transition_probabilities(state, action)
                action_values[action] = sum(prob * (rewards.get(next_state, 0) + gamma * V.ge
            best_action = max(action_values, key=action_values.get)
            policy[state] = best_action
            if old_action != best_action:
                policy_stable = False
    return policy_stable

theta = 0.0001

while True:
    V = policy_evaluation(policy, states, rewards, gamma, theta)
    if policy_improvement(policy, V, states, rewards, gamma):
        break

print("\nOptimal Policy:")
for state, action in policy.items():
    print(f"State {state}: {action}")
print("\nOptimal Value Function:")
for state, value in V.items():
    print(f"State {state}: {value:.2f}")
```

```
Iteration 0 - Value Matrix:
[[0.    0.    1.    0.   ]
 [0.    0.    0.9   0.   ]
 [0.    0.    0.81  0.729]]
Iteration 1 - Value Matrix:
[[0.    0.9   1.    0.   ]
 [0.    0.    0.9   0.   ]
 [0.    0.729 0.81  0.729]]
Iteration 2 - Value Matrix:
[[0.81   0.9    1.     0.    ]
 [0.729  0.     0.9    0.    ]
 [0.6561 0.729  0.81   0.729 ]]

Optimal Policy:
State (0, 0): Right
State (0, 1): Right
State (0, 2): Right
State (1, 0): Up
State (1, 2): Up
State (2, 0): Up
State (2, 1): Right
State (2, 2): Up
State (2, 3): Left

Optimal Value Function:
State (0, 0): 0.81
State (0, 1): 0.90
State (0, 2): 1.00
State (0, 3): 0.00
State (1, 0): 0.73
State (1, 2): 0.90
State (1, 3): 0.00
State (2, 0): 0.66
State (2, 1): 0.73
State (2, 2): 0.81
State (2, 3): 0.73
```

EXPERIMENT NO. 7

Student Name and Roll Number: Piyush Gambir
Semester /Section: Semester V – AIML B
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL348-RL-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 26. 09.2023
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.
   A Markov Decision Process is a mathematical model used in reinforcement learning, where an agent makes decisions in an environment to maximize a cumulative reward. It is defined by a set of states, a set of actions, transition probabilities, rewards, and a discount factor.

2. Can we obtain the optimal policy using value iteration algorithm?
   The value iteration algorithm can be used to obtain the optimal policy. By iteratively updating the value function until it converges to the optimal values, the corresponding greedy policy can be extracted from the optimal value function.

3. Compare and contrast policy and value iteration algorithms.
   - Policy Iteration:
     Steps: Involves alternating between policy evaluation and policy improvement until convergence.
     Policy Update: The policy is updated in a greedy manner based on the current value function.
     Convergence: Generally, converges in more iterations but each iteration may be computationally cheaper.
   - Value Iteration:
     Steps: Combines policy evaluation and policy improvement in a single step, iteratively updating the value function.
     Policy Update: The policy is implicitly updated by choosing actions greedily based on the current value function.

Convergence: Tends to converge faster per iteration but each iteration may be computationally more expensive.

- Commonality: Both algorithms converge to the optimal policy and value function in the limit.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Problem Statement:

Write a python program to implement value iteration in dynamic programming.

Code

In [ ]:
```python
# importing required libraries
import numpy as np
```

In [ ]:
```python
def compute_state_value(state, policy, V, gamma, transition_model):
    new_v = 0
    for next_state, reward, probability in transition_model.get((state, policy[state]), []):
        new_v += probability * (reward + gamma * V[next_state])
    return new_v
```

In [ ]:
```python
def policy_evaluation(V, policy, gamma, transition_model):
    while True:
        delta = 0
        for state in V:
            v = V[state]
            new_v = compute_state_value(state, policy, V, gamma, transition_model)
            V[state] = new_v
            delta = max(delta, abs(v - new_v))
        if delta < 1e-6:
            break
```

In [ ]:
```python
def policy_improvement(V, policy, gamma, transition_model):
    policy_stable = True
    for state in V:
        old_action = policy[state]
        action_values = {}
        for action in transition_model.keys():
            action_values[action] = compute_state_value(state, {state: action}, V, gamma, tra
        best_action = max(action_values, key=action_values.get)
        policy[state] = best_action
        if old_action != best_action:
            policy_stable = False
    return policy_stable
```

In [ ]:
```python
if __name__ == "__main__":
    transition_model = {
        (0, 'A'): [(0, 10, 0.8), (1, -10, 0.2)],
        (0, 'B'): [(1, 0, 1.0)],
        (1, 'A'): [(0, 0, 1.0)],
        (1, 'B'): [(1, 0, 1.0)]
    }

    policy = {0: 'A', 1: 'A'}
    V = {0: 0, 1: 0}
    gamma = 0.9

    while True:
        policy_evaluation(V, policy, gamma, transition_model)
        policy_stable = policy_improvement(V, policy, gamma, transition_model)
        if policy_stable:
```

```
            break

    print("Optimal Policy:")
    for state, action in policy.items():
        print(f"State {state}: Action {action}")
```

```
Optimal Policy:
State 0: Action (0, 'A')
State 1: Action (0, 'A')
```

EXPERIMENT NO. 8

Student Name and Roll Number: Piyush Gambir
Semester /Section: Semester V – AIML B
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL348-RL-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 17. 10.2023
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. • Policy-Based RL: Involves learning a policy that directly maps states to actions. The focus is on finding the optimal policy that maximizes expected cumulative rewards. • Value-Based RL: Focuses on estimating the value of each state or state-action pair. It aims to find the optimal value function, which represents the expected cumulative reward from a given state (or state-action pair). **2.** What are off-policy and on-policy learners? • Off-Policy Learning: Learners evaluate or improve a policy that is different from the one generating the data. Q-learning is an example of an off-policy algorithm. • On-Policy Learning: Learners optimize the policy that is used to generate the data. SARSA is an example of an on-policy algorithm. **3.** What is the Bellman equation?

The Bellman equation expresses the relationship between the value of a state (or state-action pair) and the expected immediate reward plus the discounted value of the next state (or next state-action pair).

4. What will be the effect(s) of changing the learning rate in Q-Learning?

- Too High Learning Rate: Convergence issues may arise, and the algorithm might fail to converge to an optimal solution. It may oscillate or diverge.
- Too Low Learning Rate: Slow convergence, and the algorithm may take a long time to learn or may get stuck in local optima.
- Optimal Learning Rate: Balances the trade-off between exploration and exploitation, leading to stable convergence and faster learning. The optimal learning rate depends on the problem and needs to be tuned experimentally.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Problem Statement:

Implement Q-Learning using Python

Code

In [ ]:
```
# importing required libraries
import numpy as np
```

In [ ]:
```
states = 6  # S0, S1, S2, S3, S4, S5
actions = 6  # The possible actions are to move to any of the 6 states
gamma = 0.9  # Discount factor

# Initializing the Q-table with Zeroes
Q = np.zeros((states, actions))
print("Initial Q-table")
print(Q)
print()

# Intializing the Reward Table
R = np.full((states, actions), -1)  # Default reward is -1
# Reward for Reaching Goal State is 100
R[1, 5] = R[4, 5] = R[5, 5] = 100
# Reward for Path Exising But Not Goal State is 0
R[0, 4] = R[1, 3] = R[2, 3] = R[3, 1] = R[3, 2] = R[3,
                                                    4] = R[4, 0] = R[4, 3] = R[5, 1] = R[5, 4

print("Reward Table")
print(R)
print()

episodes_paths = [
    [1, 3, 4, 5],
    [0, 4, 5],
    [4, 0, 4, 5]
]

# Function to update the Q-value


def update_q(state, action, reward, next_state):
    max_next_q = np.max(Q[next_state, :][Q[next_state, :] >= 0])
    Q[state, action] = reward + gamma * max_next_q
    print(f"Q[{state}, {action}]: {Q[state, action]}")


# Q Learning Algorithm
for episode, path in enumerate(episodes_paths):
    print(f"Episode {episode + 1} Path: {path}")
    for i in range(len(path)-1):
        state = path[i]
        next_state = path[i+1]
        action = next_state
        reward = R[state, action]
        print(f"State: {state}, Action: {action}, Reward: {reward}")
        update_q(state, action, reward, next_state)
```

```
    # Printing the Q-table
    print(f"Episode {episode + 1} Q-table:")
    print(Q)
    print()
```

```
Initial Q-table
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]

Reward Table
[[ -1  -1  -1  -1   0  -1]
 [ -1  -1  -1   0  -1 100]
 [ -1  -1  -1   0  -1  -1]
 [ -1   0   0  -1   0  -1]
 [  0  -1  -1   0  -1 100]
 [ -1   0  -1  -1   0 100]]

Episode 1 Path: [1, 3, 4, 5]
State: 1, Action: 3, Reward: 0
Q[1, 3]: 0.0
State: 3, Action: 4, Reward: 0
Q[3, 4]: 0.0
State: 4, Action: 5, Reward: 100
Q[4, 5]: 100.0
Episode 1 Q-table:
[[  0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0. 100.]
 [  0.   0.   0.   0.   0.   0.]]

Episode 2 Path: [0, 4, 5]
State: 0, Action: 4, Reward: 0
Q[0, 4]: 90.0
State: 4, Action: 5, Reward: 100
Q[4, 5]: 100.0
Episode 2 Q-table:
[[  0.   0.   0.   0.  90.   0.]
 [  0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0. 100.]
 [  0.   0.   0.   0.   0.   0.]]

Episode 3 Path: [4, 0, 4, 5]
State: 4, Action: 0, Reward: 0
Q[4, 0]: 81.0
State: 0, Action: 4, Reward: 0
Q[0, 4]: 90.0
State: 4, Action: 5, Reward: 100
Q[4, 5]: 100.0
Episode 3 Q-table:
[[  0.   0.   0.   0.  90.   0.]
 [  0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.]
 [ 81.   0.   0.   0.   0. 100.]
 [  0.   0.   0.   0.   0.   0.]]
```

EXPERIMENT NO. 9

Student Name and Roll Number: Piyush Gambir
Semester /Section: Semester V – AIML B
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL348-RL-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 31. 10.2023
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 Algorithm.

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?

   Episodic Markov Decision Processes (MDPs) are a type of MDP where an agent's interaction with the environment is divided into episodes. Each episode consists of a sequence of states, actions, and rewards, starting from an initial state and ending in a terminal state. The agent's goal is to learn a policy that maximizes the expected cumulative reward over each episode.

2. What are model-free and model-based methods in RL?
   - Model-Free Methods: These approaches in reinforcement learning (RL) do not require knowledge of the underlying dynamics of the environment. Instead, they focus on learning a policy or a value function directly from the observed experiences (trajectories) of the agent.

- Model-Based Methods: In contrast, model-based methods involve building a model of the environment's dynamics. The agent then uses this model to simulate possible outcomes and plan its actions accordingly. Model-based methods can be more sample-efficient but rely on an accurate model of the environment.

3. Differentiate between on-policy and off-policy learning in RL.
   - On-Policy Learning: In on-policy learning, the agent learns and updates its policy based on the data generated by its current policy. The data used for learning comes from the same policy that is being improved. Examples include SARSA and REINFORCE algorithms.
   - Off-Policy Learning: Off-policy learning allows the agent to learn from data generated by a different policy. The agent can learn from historical data or different behavioral policies, making it more flexible. Q-learning is an example of an off-policy algorithm.

4. What are exploring starts in Monte Carlo?
   In the context of Monte Carlo methods in reinforcement learning, exploring starts refers to a strategy where episodes begin with a randomly chosen initial state and action pair. This is used in Monte Carlo control methods to ensure that all state-action pairs have a non-zero probability of being visited, contributing to a more comprehensive exploration of the state space.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Problem Statement:

Implement Monte Carlo Algorithm.

Code

In [ ]:
```python
# importing required libraries
import numpy as np
```

First Visit

In [ ]:
```python
def monte_carlo_first_visit(episodes):
    returns = {}
    state_count = {}
    state_values = {}

    for episode in episodes:
        states, rewards = zip(*episode)
        total_return = 0

        for t in range(len(states) - 1, -1, -1):
            state = states[t]
            total_return += rewards[t]
            if state not in states[:t]:
                if state in returns:
                    returns[state].append(total_return)
                else:
                    returns[state] = [total_return]
                state_count[state] = len(returns[state])
                state_values[state] = sum(returns[state]) / state_count[state]

    return state_values


if __name__ == "__main__":
    num_episodes = 2
    episodes = [
        [('A', 3), ('A', 2), ('B', -4), ('A', 4), ('B', -3)],
        [('B', -2), ('A', 3), ('B', -3)],
    ]

    state_values = monte_carlo_first_visit(episodes)

    print("Episodes")
    i = 0
    for episode in episodes:
        i += 1
        print("Episode:" + str(i))
        for state, reward in episode:
            print(f"State {state}: Reward {reward}")
        print("")

    print("Estimated state values:")
    for state, value in state_values.items():
        print(f"State {state}: {value}")
```

```
Episodes
Episode:1
State A: Reward 3
State A: Reward 2
State B: Reward -4
State A: Reward 4
State B: Reward -3

Episode:2
State B: Reward -2
State A: Reward 3
State B: Reward -3

Estimated state values:
State B: -2.5
State A: 1.0
```

Every Visit

In [ ]:

```python
gamma = 1.0


def calculate_return(episode, t):
    G = 0
    for i in range(t, len(episode)):
        reward = episode[i][1]
        G = G + (gamma ** (i - t)) * reward
    return G


def monte_carlo_every_visit(episodes):
    state_values = {}
    returns_sum = {}
    state_counts = {}

    for episode in episodes:
        for t in range(len(episode)):
            state = episode[t][0]
            if state not in state_counts:
                state_counts[state] = 0
            state_counts[state] += 1

            G = calculate_return(episode, t)

            if state not in returns_sum:
                returns_sum[state] = 0
            returns_sum[state] += G

            state_values[state] = returns_sum[state] / state_counts[state]

    return state_values


if __name__ == "__main__":
    episodes = [
        [('A', 3), ('A', 2), ('B', -4), ('A', 4), ('B', -3)],
        [('B', -2), ('A', 3), ('B', -3)],
    ]
    state_values = monte_carlo_every_visit(episodes)

    print("Episodes")
    i = 0
    for episode in episodes:
        i += 1
        print("Episode:" + str(i))
```

```
        for state, reward in episode:
            print(f"State {state}: Reward {reward}")
        print("")

    print("Estimated state values:")
    for state, value in state_values.items():
        print(f"State {state}: {value}")
```

```
Episodes
Episode:1
State A: Reward 3
State A: Reward 2
State B: Reward -4
State A: Reward 4
State B: Reward -3

Episode:2
State B: Reward -2
State A: Reward 3
State B: Reward -3

Estimated state values:
State A: 0.5
State B: -2.75
```

EXPERIMENT NO. 10 (Value Added Experiment)

Student Name and Roll Number: Piyush Gambir
Semester /Section: Semester V – AIML B
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL348-RL-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 14. 11.2023
Faculty Signature:
Marks/Grade:

Objective(s): To understand Q Learning and Frozen Lake Environment and apply in Reinforcement Learning scenarios.
Outcome: Students will be familiarized with Q Learning and OpenAI Gym Library.
Problem Statement: Write Python Program to implement Q Learning on the Frozen Lake Environment.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Experiment 10

Problem Statement

Write a program to implement Q-Learning on Frozen Lake Environment.

Installing Dependencies

In [ ]:
```
! pip install gymnasium
```

```
Requirement already satisfied: gymnasium in c:\users\mainp\appdata\local\programs\python\pytho
n311\lib\site-packages (0.28.1)
Requirement already satisfied: numpy>=1.21.0 in c:\users\mainp\appdata\local\programs\python\p
ython311\lib\site-packages (from gymnasium) (1.25.0)
Requirement already satisfied: jax-jumpy>=1.0.0 in c:\users\mainp\appdata\local\programs\pytho
n\python311\lib\site-packages (from gymnasium) (1.0.0)
Requirement already satisfied: cloudpickle>=1.2.0 in c:\users\mainp\appdata\local\programs\pyt
hon\python311\lib\site-packages (from gymnasium) (2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in c:\users\mainp\appdata\local\progra
ms\python\python311\lib\site-packages (from gymnasium) (4.6.3)
Requirement already satisfied: farama-notifications>=0.0.1 in c:\users\mainp\appdata\local\pro
grams\python\python311\lib\site-packages (from gymnasium) (0.0.4)
```

Code:

Importing the Required Libraries

In [ ]:
```
# importing the required libraries
import numpy as np
import matplotlib.pyplot as plt
import pickle
import gymnasium as gym
from gymnasium.envs.toy_text.frozen_lake import generate_random_map
```

In [ ]:
```
training = True
render = False
```

Creating the Environment

In [ ]:
```
# creating the frozen lake environment
frozen_lake_environment = gym.make(
    'FrozenLake-v1', map_name="4x4", is_slippery=True, render_mode='human' if render else Non
```

In [ ]:
```
# action space of the frozen lake environment
environment_action_space = frozen_lake_environment.action_space
print("Action space:", environment_action_space)

# observation space of the frozen lake environment
environment_observation_space = frozen_lake_environment.observation_space
print("Observation space:", environment_observation_space)
```

```
Action space: Discrete(4)
Observation space: Discrete(16)
```

Implementing the Q-learning Algorithm

Creating/Loading the Q-table

In [ ]:
```
if (training):
    # creating the Q table for the frozen lake environment
    q_table = np.zeros((environment_observation_space.n,
                        environment_action_space.n))
    print("Q table size:", q_table.shape)
    print("Q table:", q_table)
else:
    # loading the Q table from the file
    q_table = pickle.load(open("q_table.pkl", "rb"))
    print("Q table size:", q_table.shape)
    print("Q table:", q_table)
```

```
Q table size: (16, 4)
Q table: [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Setting the Hyperparameters

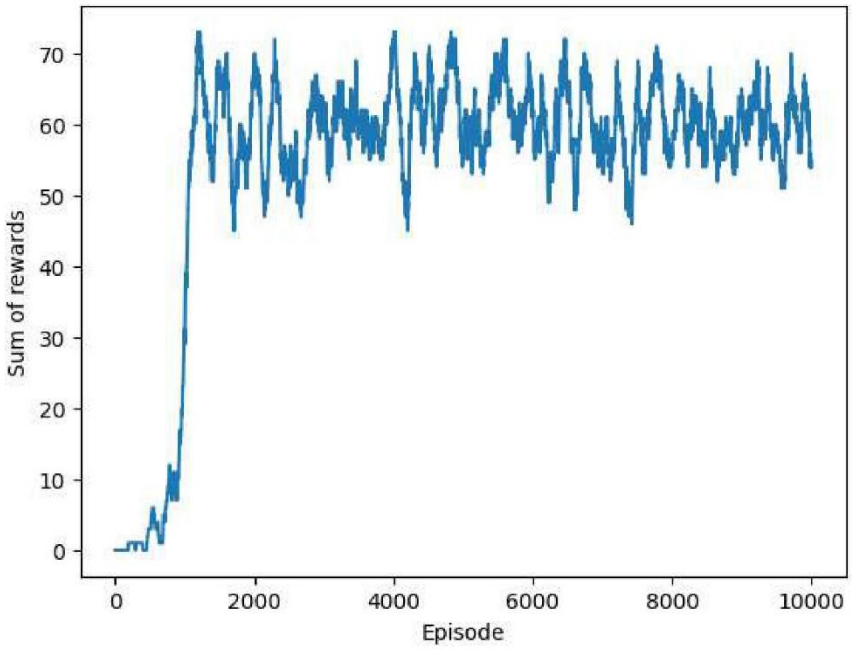
In [ ]:
```
# specifying the hyperparameters

# number of episodes
total_episodes = 10000

# maximum number of steps per episode
maximum_steps_per_episode = 100

# learning rate
learning_rate = 0.9

# discounting rate
discounting_rate = 0.9

# exploration rate
exploration_rate = 1.0

# maximum exploration rate
maximum_exploration_rate = 1.0

# minimum exploration rate
minimum_exploration_rate = 0.01

# exploration rate decay
exploration_rate_decay = 0.001
```

Applying the Q-learning Algorithm

In [ ]:
```
rewards_per_episode = np.zeros(total_episodes)
for i in range(total_episodes):
    # reset the environment to get the initial state
    state = frozen_lake_environment.reset()[0]
    terminal_state = False  # initialize the terminal state to False
    truncated = False  # initialize the truncate to False

    while (not terminal_state and not truncated):
        if (training and np.random.random() < exploration_rate):
            # choose an action randomly
            action = frozen_lake_environment.action_space.sample()
        else:
            # choose the action with the highest Q value for the current state
            action = np.argmax(q_table[state, :])

        # choose an action

        new_state, reward, terminal_state, truncated, prob = frozen_lake_environment.step(
            action)

        if (training):
            q_table[state, action] = q_table[state, action] + learning_rate * \
                (reward + discounting_rate *
                 np.max(q_table[new_state, :]) - q_table[state, action])

        state = new_state

    exploration_rate = max(exploration_rate - exploration_rate_decay, 0)

    if (exploration_rate == 0):
        learning_rate = 0.0001

    if (reward == 1):
        rewards_per_episode[i] = 1

frozen_lake_environment.close()
```

In [ ]:
```
# save the Q table if training
if (training):
    with open("q_table.pkl", "wb") as file:
        pickle.dump(q_table, file)
```

In [ ]:
```
sum_of_rewards = np.zeros(total_episodes)
for i in range(total_episodes):
    sum_of_rewards[i] = np.sum(rewards_per_episode[max(0, i-100):i+1])

plt.plot(sum_of_rewards)
plt.xlabel("Episode")
plt.ylabel("Sum of rewards")
plt.show()
```


Sum of rewards
Episode
0
10
20
30
40
50
60
70
0
2000
4000
6000
8000
10000