EXPERIMENT NO. 4

Student Name and Roll Number: Piyush Gambhir – 21CSU349
Semester /Section: Semester-V – AIML-V-B (AL-3)
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL347-AAIES-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 02.09.2023
Faculty Signature:
Grade:

Objective(s):

- Understand what Greedy Algorithm is.
- Study about different algorithm design paradigms.
- Implement greedy for solving a real-world problem

Outcome:

Students will be familiarized with Greedy Algorithm
Students will be able to make a comparison between the two algorithms.

Problem Statement:

Write a program for the following problem:

A company is planning to launch a new product. They have a limited budget to spend on marketing and advertising. They need to decide how to allocate their budget to maximize the number of people who will be aware of their product.

Marketing Channels:

Social Media: Cost - $50, Reach - 1000 people aware of the product.
Email Campaign: Cost - $80, Reach - 1500 people aware of the product.
Influencer Collaboration: Cost - $120, Reach - 2500 people aware of the product.

Budget Constraint: $200

Now, the company wants to allocate their budget to these marketing channels in such a way that they maximize the total number of people aware of their product.

Background Study:

A greedy algorithm is a heuristic-based technique used in problem-solving. It makes locally optimal choices at each step, hoping that those choices will lead to a global optimal solution. The algorithm selects the best option available at the current state without revisiting or undoing its decisions.

Question Bank:

1. How can you solve a problem using a Greedy approach?

A Greedy approach involves making locally optimal choices at each step to find the global optimal solution. In a greedy algorithm, you make the best choice available at the moment without considering the consequences of that choice on future steps. Greedy algorithms are useful for optimization problems where finding an exact solution might be complex or time-consuming. While a Greedy approach doesn't guarantee the optimal solution for all problems, it can work well for problems where the greedy choice is always the best choice and doesn't lead to incorrect results.

2. What are the advantages and disadvantages of Greedy algorithm.

Advantages:

- Greedy algorithms are often easy to understand and implement.
- They can be efficient and provide quick solutions for some optimization problems.
- Greedy algorithms are useful when the problem has a greedy property, where making locally optimal choices leads to a globally optimal solution.

Disadvantages:

- Greedy algorithms do not guarantee finding the globally optimal solution in all cases. They might lead to suboptimal solutions.
- The greedy choice that seems best at the moment might not be the best choice for achieving the overall optimal solution.
- Determining whether a problem can be solved optimally using a greedy approach can be challenging.
- Greedy algorithms might require additional verification steps to ensure the solution's correctness.

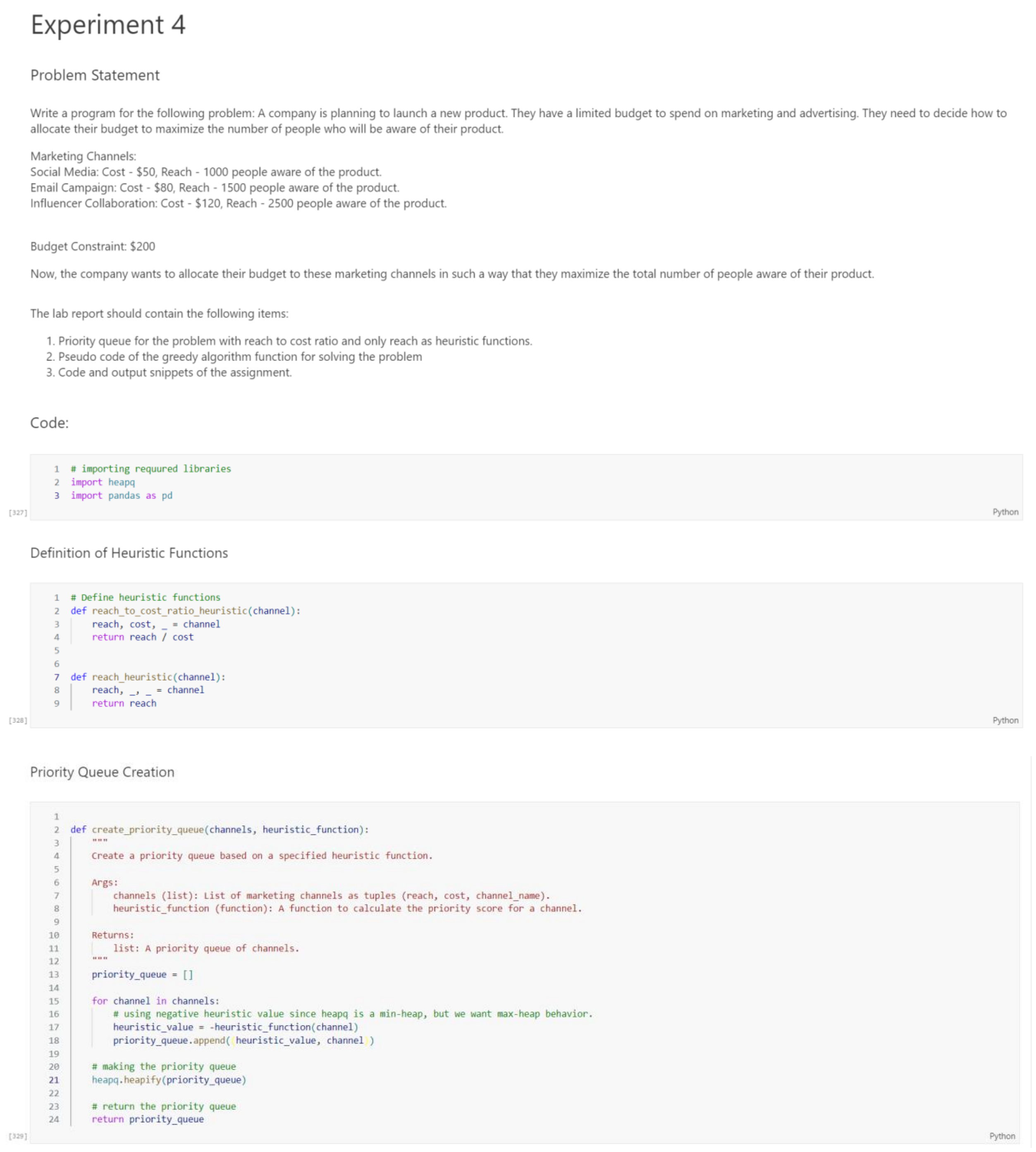
Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Pseudocode:

```
function reach_to_cost_ratio_heuristic(channel):
    reach, cost, _ = channel
    return reach / cost

function reach_heuristic(channel):
    reach, _, _ = channel
    return reach

function create_priority_queue(channels, heuristic_function):
    priority_queue = []
    for channel in channels:
        heuristic_value = -heuristic_function(channel)
        priority_queue.append((heuristic_value, channel))
    heapq.heapify(priority_queue)
    return priority_queue

function greedy_allocation(priority_queue, budget):
    allocated_channels = []
    remaining_budget = budget
    total_cost_used = 0
    total_reach = 0
    while priority_queue and remaining_budget > 0:
        _, channel = heapq.heappop(priority_queue)
        reach, cost, channel_name = channel
        if priority_queue and cost <= remaining_budget:
            allocated_channels.append((channel_name, cost))
            remaining_budget -= cost
            total_cost_used += cost
            total_reach += reach
    return allocated_channels, total_cost_used, total_reach


function main():
    social_media = (1000, 50, "Social Media")
    email_campaign = (1500, 80, "Email Campaign")
    influencer_collaboration = (2500, 120, "Influencer Collaboration")
    marketing_channels = [social_media, email_campaign, influencer_collaboration]
    budget_constraint = 200

    pq_reach_heuristic = create_priority_queue(marketing_channels, reach_heuristic)

    for channel in pq_reach_heuristic:
        print(channel)

    result_reach_heuristic, total_cost_reach_heuristic, total_reach_reach_heuristic = greedy_allocation(pq_reach_heuristic, budget_constraint)

    for channel_name, cost in result_reach_heuristic:
        print(f"{channel_name}: ${cost}")
    print("Total Cost Using Reach Heuristic: $", total_cost_reach_heuristic)
    print("Total Reach Using Reach Heuristic:", total_reach_reach_heuristic)

    allocation_df = create_dataframe(marketing_channels)

    pq_cost_ratio_heuristic = create_priority_queue(marketing_channels, reach_to_cost_ratio_heuristic)

    for channel in pq_cost_ratio_heuristic:
        print(channel)

    result_cost_ratio_heuristic, total_cost_cost_ratio_heuristic, total_reach_cost_ratio_heuristic = greedy_allocation(pq_cost_ratio_heuristic,
budget_constraint)

    for channel_name, cost in result_cost_ratio_heuristic:
        print(f"{channel_name}: ${cost}")
    print("Total Cost Using Cost-Ratio Heuristic: $", total_cost_cost_ratio_heuristic)
    print("Total Reach Using Cost-Ratio Heuristic:", total_reach_cost_ratio_heuristic)

function create_dataframe(marketing_channels):
    allocation_df = pd.DataFrame(
        {"Channel Name": [channel[2] for channel in marketing_channels],
         "Cost Ratio": [reach_to_cost_ratio_heuristic(channel) for channel in marketing_channels]}
    )
    return allocation_df

if __name__ == "__main__":
    main()
```

Code:



Experiment 4

Problem Statement

Write a program for the following problem: A company is planning to launch a new product. They have a limited budget to spend on marketing and advertising. They need to decide how to allocate their budget to maximize the number of people who will be aware of their product.

Marketing Channels:
Social Media: Cost - $50, Reach - 1000 people aware of the product.
Email Campaign: Cost - $80, Reach - 1500 people aware of the product.
Influencer Collaboration: Cost - $120, Reach - 2500 people aware of the product.

Budget Constraint: $200

Now, the company wants to allocate their budget to these marketing channels in such a way that they maximize the total number of people aware of their product.

The lab report should contain the following items:

1. Priority queue for the problem with reach to cost ratio and only reach as heuristic functions.
2. Pseudo code of the greedy algorithm function for solving the problem
3. Code and output snippets of the assignment.

Code:

```python
# importing requred libraries
import heapq
import pandas as pd
```

Definition of Heuristic Functions

```python
# Define heuristic functions
def reach_to_cost_ratio_heuristic(channel):
    reach, cost, _ = channel
    return reach / cost


def reach_heuristic(channel):
    reach, _, _ = channel
    return reach
```

Priority Queue Creation

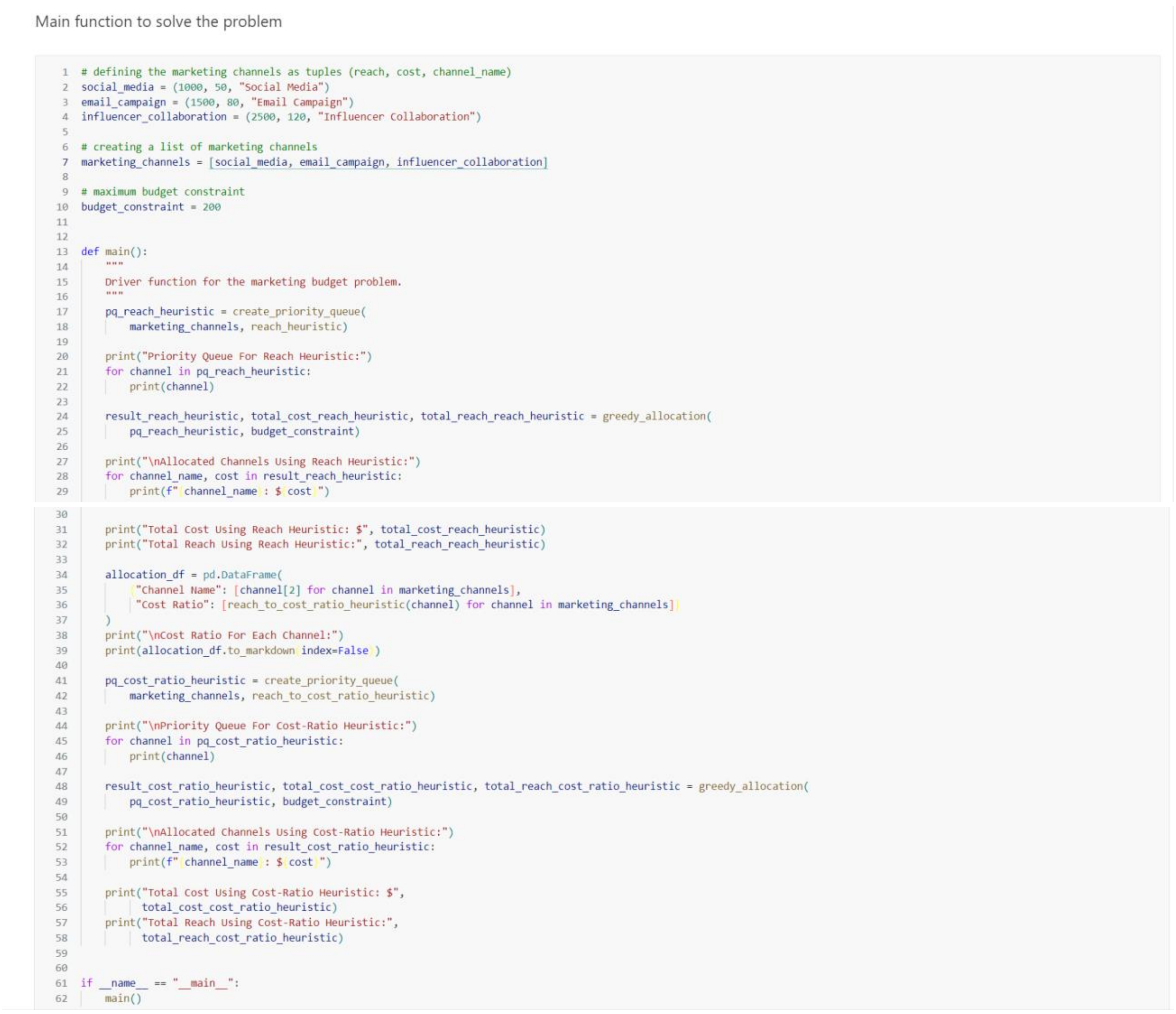
```python

def create_priority_queue(channels, heuristic_function):
    """
    Create a priority queue based on a specified heuristic function.

    Args:
        channels (list): List of marketing channels as tuples (reach, cost, channel_name).
        heuristic_function (function): A function to calculate the priority score for a channel.

    Returns:
        list: A priority queue of channels.
    """
    priority_queue = []

    for channel in channels:
        # using negative heuristic value since heapq is a min-heap, but we want max-heap behavior.
        heuristic_value = -heuristic_function(channel)
        priority_queue.append( heuristic_value, channel )

    # making the priority queue
    heapq.heapify(priority_queue)

    # return the priority queue
    return priority_queue
```

Defining the Greedy algorithm

```python
def greedy_allocation(priority_queue, budget):
    """
    Allocate budget greedily based on the priority queue.

    Args:
        priority_queue (list): A priority queue of channels.
        budget (int): The budget constraint.

    Returns:
        list: A list of allocated channels.
    """
    allocated_channels = []
    remaining_budget = budget
    total_cost_used = 0
    total_reach = 0

    while priority_queue and remaining_budget > 0:
        _, channel = heapq.heappop(priority_queue)
        reach, cost, channel_name = channel

        if priority_queue and cost <= remaining_budget:
            allocated_channels.append((channel_name, cost))
            remaining_budget -= cost
            total_cost_used += cost
            total_reach += reach

    return allocated_channels, total_cost_used, total_reach
```
[330]

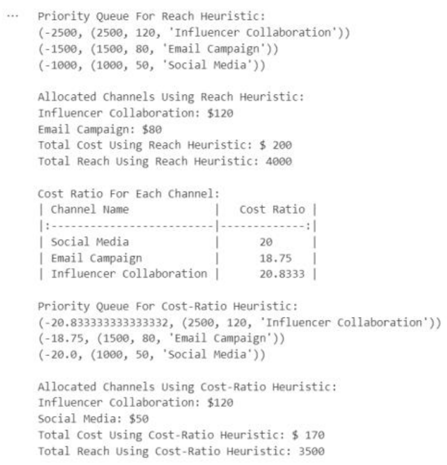
Python

Output:



Main function to solve the problem

```python
# defining the marketing channels as tuples (reach, cost, channel_name)
social_media = (1000, 50, "Social Media")
email_campaign = (1500, 80, "Email Campaign")
influencer_collaboration = (2500, 120, "Influencer Collaboration")

# creating a list of marketing channels
marketing_channels = [social_media, email_campaign, influencer_collaboration]

# maximum budget constraint
budget_constraint = 200


def main():
    """
    Driver function for the marketing budget problem.
    """
    pq_reach_heuristic = create_priority_queue(
        marketing_channels, reach_heuristic)

    print("Priority Queue For Reach Heuristic:")
    for channel in pq_reach_heuristic:
        print(channel)

    result_reach_heuristic, total_cost_reach_heuristic, total_reach_reach_heuristic = greedy_allocation(
        pq_reach_heuristic, budget_constraint)

    print("\nAllocated Channels Using Reach Heuristic:")
    for channel_name, cost in result_reach_heuristic:
        print(f"{channel_name}: ${cost}")

    print("Total Cost Using Reach Heuristic: $", total_cost_reach_heuristic)
    print("Total Reach Using Reach Heuristic:", total_reach_reach_heuristic)

    allocation_df = pd.DataFrame(
        {"Channel Name": [channel[2] for channel in marketing_channels],
         "Cost Ratio": [reach_to_cost_ratio_heuristic(channel) for channel in marketing_channels]}
    )
    print("\nCost Ratio For Each Channel:")
    print(allocation_df.to_markdown(index=False))

    pq_cost_ratio_heuristic = create_priority_queue(
        marketing_channels, reach_to_cost_ratio_heuristic)

    print("\nPriority Queue For Cost-Ratio Heuristic:")
    for channel in pq_cost_ratio_heuristic:
        print(channel)

    result_cost_ratio_heuristic, total_cost_cost_ratio_heuristic, total_reach_cost_ratio_heuristic = greedy_allocation(
        pq_cost_ratio_heuristic, budget_constraint)

    print("\nAllocated Channels Using Cost-Ratio Heuristic:")
    for channel_name, cost in result_cost_ratio_heuristic:
        print(f"{channel_name}: ${cost}")

    print("Total Cost Using Cost-Ratio Heuristic: $",
          total_cost_cost_ratio_heuristic)
    print("Total Reach Using Cost-Ratio Heuristic:",
          total_reach_cost_ratio_heuristic)


if __name__ == "__main__":
    main()
```



```
...  Priority Queue For Reach Heuristic:
     (-2500, (2500, 120, 'Influencer Collaboration'))
     (-1500, (1500, 80, 'Email Campaign'))
     (-1000, (1000, 50, 'Social Media'))

     Allocated Channels Using Reach Heuristic:
     Influencer Collaboration: $120
     Email Campaign: $80
     Total Cost Using Reach Heuristic: $ 200
     Total Reach Using Reach Heuristic: 4000

     Cost Ratio For Each Channel:
     | Channel Name             |   Cost Ratio |
     |:-------------------------|-------------:|
     | Social Media             |      20      |
     | Email Campaign           |      18.75   |
     | Influencer Collaboration |      20.8333 |

     Priority Queue For Cost-Ratio Heuristic:
     (-20.833333333333332, (2500, 120, 'Influencer Collaboration'))
     (-18.75, (1500, 80, 'Email Campaign'))
     (-20.0, (1000, 50, 'Social Media'))

     Allocated Channels Using Cost-Ratio Heuristic:
     Influencer Collaboration: $120
     Social Media: $50
     Total Cost Using Cost-Ratio Heuristic: $ 170
     Total Reach Using Cost-Ratio Heuristic: 3500
```