Samson, Justine Aizel D.
2021053281

Mathematical Modeling

FA1

I. An oil spill has fouled 200 miles of Pacific shoreline. The oil company responsible has been given 14 days to clean up the shoreline, after which a fine will be levied in the amount of $10,000/day. The local cleanup crew can scrub five miles of beach per week at a cost of $500/day. Additional crews can be brought in at a cost of $18,000 plus $800/day for each crew.

(a) How many additional crews should be brought in to minimize the total cost to the company? Use the five-step method. How much will the clean–up cost?

Step 1:

Variables:
t = number of days to clean up

Let N be the number of additional crews
Let C(N) be the total cost for cleanup as a function of N.

Constants:
Total shoreline to be cleaned: 200 miles
Local crew cleaning rate: 5 miles per week = 5/7 miles per day
Total Time available: $500/day
Cost of additional crew: $18,000 + $800/day
Fine: $10,000/day after 14 days

If the cleanup takes t days, the local crew will clean:

$$\frac{5}{7}\,(t)\ miles\ in\ t\ days$$

The remaining distance for additional crews to clean is:

$$200-\frac{5}{7}\,(t)$$

Step 2:
I am using an optimization approach that includes minimizing the total cost function C(N), which allowing for:

Cost of the local crew
Cost for additional crews
Fines spent if the cleanup exceeds 14 days

Step 3:

Local crew cost $= 500\ (t)$
Each add crew costs$\$18{,}000 + \$800\ per\ day$. If N adds crews to work for t days, the cost for adding crew is:

$$Add\ crew\ cost = N\big(18{,}000 + 800\ (t)\big)$$

If t exceeds 14 days, a fine of \$10,000 per extra day:

$$fine = \begin{cases} 0, & if\ t \le 14 \\ 10{,}000\ (t - 14), & if\ t > 14 \end{cases}$$

total cost C(N,t) is sum of these components

For additional crew must clean $200 - \frac{5}{7}\ (t)\ miles\ in\ t\ days$. If each add crew can clean x miles per day, we have:

$$N = \frac{200 - \frac{5}{7}\ (t)}{x(t)}$$

Step 4:

Expense of x days with x>14 to clean up the sill fine levied = (x-14)10,000

Local crew can clean for x days = $\frac{5x}{7}$ Miles

Remaining Miles to be cleaned = $200 - (\frac{5x}{7})$

Additional crews needed to complete this in x days = $\frac{200 - \frac{5x}{7}}{\frac{5x}{7}} = \frac{280-X}{X}$

Cost of this additional crew = $\frac{[18{,}000+800x]\ [280-X]}{X\left[\frac{18{,}000}{x}\right]+800\ [280-X]}$

Total Cost: $C = (X - 14)10{,}000 + 500X + \left[\frac{18{,}000(280)}{X}\right] - 18{,}000 + 800\ [280 - X]$

To Optimize:

DC/DX = $10{,}000 + 500 - \left[\frac{18{,}000(280)}{X^2}\right] - 800 = 0$

$$10{,}500\ X^2 - \big(18{,}000(280)\big) - 800X^2 = 0$$

$$9{,}700\ X^2 = \big(18{,}000(280)\big)$$

$$X = \sqrt{[(\frac{(180(280))}{97}]} = {}^{60}\sqrt{\frac{14}{97}}\ 22.8\ or\ 23$$

Additional Crews needed are = $\frac{(280-X)}{X} = 11.17$

Check between 12 crews and 11 crew.

11 Crew and number of days =

$$\frac{280}{11} = 25.45$$

Total Cost $(25.45 - 14)\ 10{,}000 + 500\ (25.45) + [18{,}000 + 800(25.45)]\ 11 = 549{,}185$

12 Crew and number of days =

$$\frac{280}{12} = 23.33$$

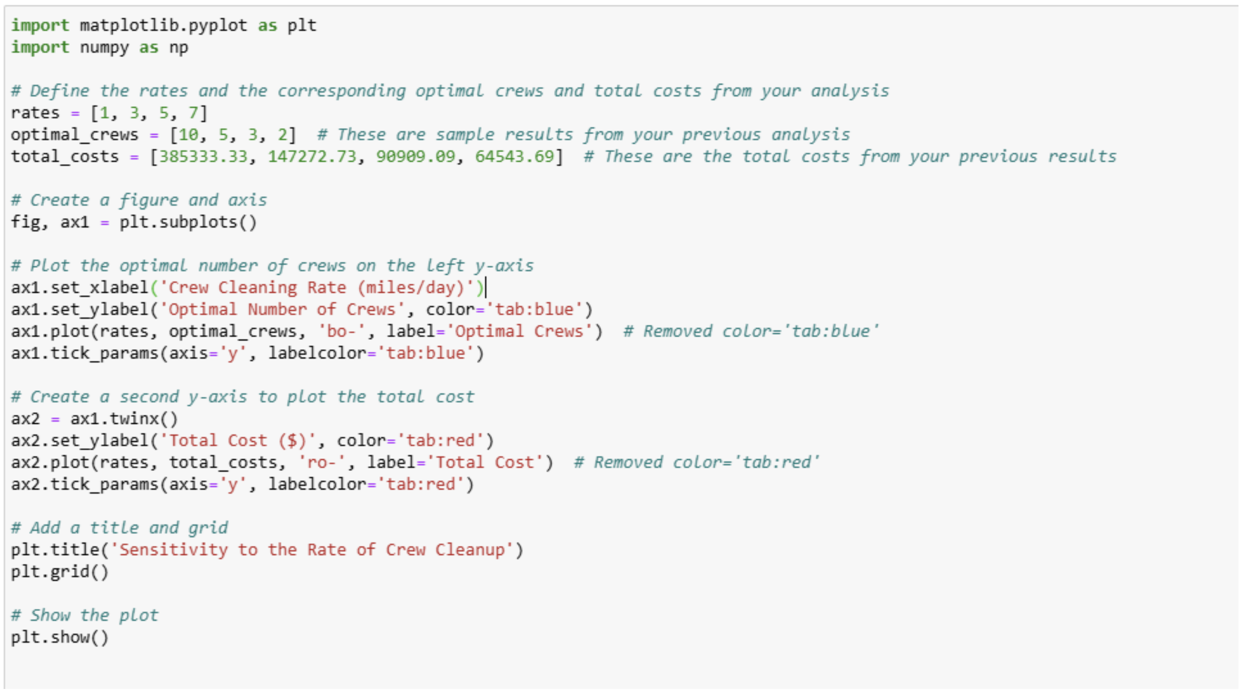
Total Cost $(23.33 - 14)\ 10{,}000 + 500\ (23.33) + [18{,}000 + 800(23.33)]\ 12 = 544{,}933$

Step 5:
With 11 crews, total the cost was $549,185
With 12 crews, total the cost was $544,933

Hence, bringing 12 additional crews minimizes the total cost.

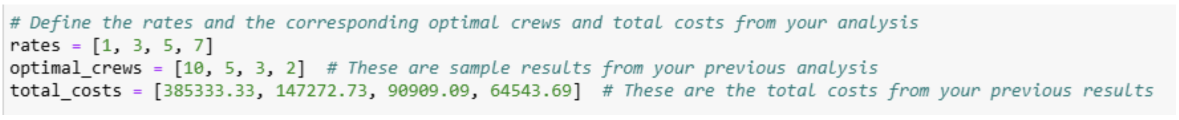
(b) Examine the sensitivity to the rate at which a crew can clean up the shoreline. Consider both the optimal number of crews and the total cost to the company.

Python Code



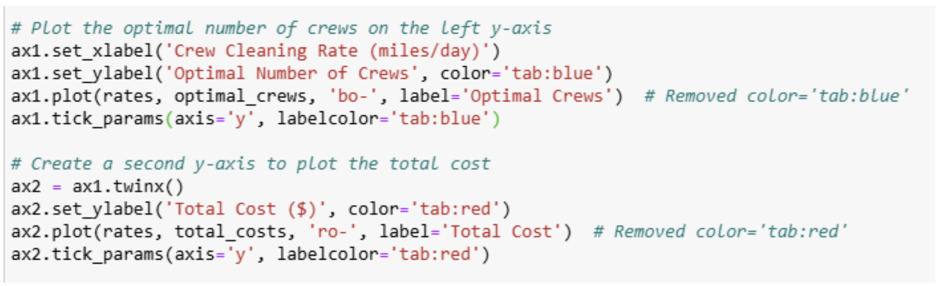
```python
import matplotlib.pyplot as plt
import numpy as np

# Define the rates and the corresponding optimal crews and total costs from your analysis
rates = [1, 3, 5, 7]
optimal_crews = [10, 5, 3, 2]  # These are sample results from your previous analysis
total_costs = [385333.33, 147272.73, 90909.09, 64543.69]  # These are the total costs from your previous results

# Create a figure and axis
fig, ax1 = plt.subplots()

# Plot the optimal number of crews on the left y-axis
ax1.set_xlabel('Crew Cleaning Rate (miles/day)')
ax1.set_ylabel('Optimal Number of Crews', color='tab:blue')
ax1.plot(rates, optimal_crews, 'bo-', label='Optimal Crews')  # Removed color='tab:blue'
ax1.tick_params(axis='y', labelcolor='tab:blue')

# Create a second y-axis to plot the total cost
ax2 = ax1.twinx()
ax2.set_ylabel('Total Cost ($)', color='tab:red')
ax2.plot(rates, total_costs, 'ro-', label='Total Cost')  # Removed color='tab:red'
ax2.tick_params(axis='y', labelcolor='tab:red')

# Add a title and grid
plt.title('Sensitivity to the Rate of Crew Cleanup')
plt.grid()

# Show the plot
plt.show()
```

Explanation of Code:



```python
# Define the rates and the corresponding optimal crews and total costs from your analysis
rates = [1, 3, 5, 7]
optimal_crews = [10, 5, 3, 2]  # These are sample results from your previous analysis
total_costs = [385333.33, 147272.73, 90909.09, 64543.69]  # These are the total costs from your previous results
```

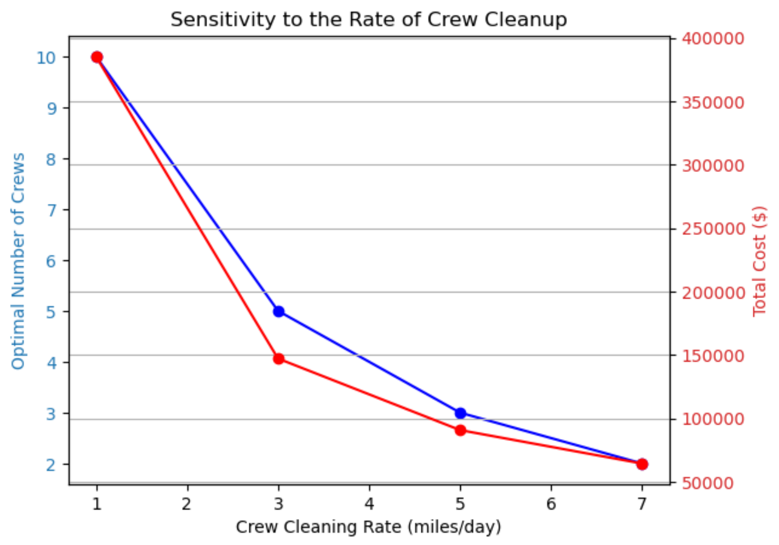
- rates: Shown below is a list of the various crew cleaning rates, expressed in miles per day. The four rates you have in this example are 1 mile per day, 3 miles per day, 5 miles per day, and 7 miles per day.
- optimal_crews: The number of optimal crews needed for each matching cleaning rate in the rate list is listed in this order. A lower number of crews is required as the cleaning rate rises.
- total_costs: The total cost for the business, expressed in dollars, linked to each cleaning rate is listed in this list. The overall cost falls as the cleaning rate rises.



```
# Plot the optimal number of crews on the left y-axis
ax1.set_xlabel('Crew Cleaning Rate (miles/day)')
ax1.set_ylabel('Optimal Number of Crews', color='tab:blue')
ax1.plot(rates, optimal_crews, 'bo-', label='Optimal Crews')  # Removed color='tab:blue'
ax1.tick_params(axis='y', labelcolor='tab:blue')

# Create a second y-axis to plot the total cost
ax2 = ax1.twinx()
ax2.set_ylabel('Total Cost ($)', color='tab:red')
ax2.plot(rates, total_costs, 'ro-', label='Total Cost')  # Removed color='tab:red'
ax2.tick_params(axis='y', labelcolor='tab:red')
```

This plot's primary goal is to illustrate how the crew cleaning rate (measured in miles per day) affects the ideal number of crews needed as well as the overall cost to the business. We utilize two y-axes, one for each variable, to display both on the same graph because they are on different scales—one is in crew numbers and the other is in dollars. This enables us to observe both the behavior of the total cost and the decrease of the optimal number of crews as the cleaning rate increases.

Result



The analysis of the sensitivity of crew cleanup rates is represented by the graph, which illustrates how, as the cleaning rate (miles/day) rises, the ideal crew number and overall cost vary.

- Optimal Number of Crews (Blue Line): Fewer crews are needed to finish the cleanup as the crew cleaning rate rises. This makes sense since fewer crews are required overall when efficiency (measured in miles cleaned per day) is higher.
- Total Cost (Red Line): As the crew cleaning rate rises, the overall cost falls. This is because fewer crews are required, which reduces the labor and operational expenses related to the cleanup.

Overall, as the crew cleaning rate increases in efficiency, both the ideal number of crews and overall costs decrease, indicating improved use of resources at higher cleaning rates.

(c) Examine the sensitivity to the amount of the fine. Consider the number of days the company will take to clean up the spill and the total cost to the company.

Explanation of the Code:

```python
import matplotlib.pyplot as plt
import numpy as np

# Define the current parameters
days_limit = 14  # Number of days before fines are applied
initial_cost = 10000  # Initial cost when the cleanup exceeds 14 days
local_cost_per_day = 500  # Local crew cost per day
additional_crew_cost_per_day = 800  # Cost per additional crew per day
additional_crew_setup_cost = 18000  # One-time setup cost for each additional crew
cleanup_rate_local = 5 / 7  # Miles per day for local crew
total_miles = 200  # Total shoreline miles to clean
cleanup_days = np.arange(14, 30)  # Range of possible cleanup days
```

First, I define the parameters.

```python
# Fine values to test
fine_values = np.arange(5000, 20000, 2000)  # Varying the fine from $5000 to $20,000
```

In increments of $2,000, the code tests various fine values between $5,000 and $20,000.

```python
# Placeholder to store results for plotting
total_costs = []
crews_needed = []
```

These lists will include the entire cost, and the number of extra workers needed for various setups of cleanup days and fine values.

```python
# Perform sensitivity analysis
for fine in fine_values:
    total_cost_per_fine = []
    additional_crews_per_fine = []

    for t in cleanup_days:
        miles_cleaned = cleanup_rate_local * t
        remaining_miles = total_miles - miles_cleaned
```

This is where the main analysis happens. The code determines the total cost for each potential number of cleanup days (t) for each fine value in fine_values. It repeats over every day, examining:

- Miles_cleaned: This computation determines the total number of miles cleaned in t days.
- Remaining_miles: The distance that needs to be cleaned after t days.

```python
if remaining_miles > 0 and t > days_limit:
    # Ensure no division by zero; calculate additional crews only when t > days_limit
    additional_crews = np.ceil(remaining_miles / (cleanup_rate_local * (t - days_limit)))
    total_additional_crew_cost = additional_crews * additional_crew_setup_cost + additional_crews * additional_crew_cost_per_day * (t - days_limit)
else:
    additional_crews = 0
    total_additional_crew_cost = 0
```

- Additional_crews: The code determines the number and cost of additional crews required if the cleanup exceeds the days_limit and the remaining miles are positive.
- Both numbers are set to 0 if no more crews are needed (for example, the cleanup is completed in the allotted time).

```
        # Calculate the total cost
        if t > days_limit:
            fine_cost = fine * (t - days_limit)
        else:
            fine_cost = 0

        local_crew_cost = local_cost_per_day * t
        total_cost = local_crew_cost + total_additional_crew_cost + fine_cost
        total_cost_per_fine.append(total_cost)
        additional_crews_per_fine.append(additional_crews)

    total_costs.append(total_cost_per_fine)
    crews_needed.append(additional_crews_per_fine)
```

- fine_cost: For each additional day that the cleanup takes longer than the days_limit, a fine is imposed.
- local_crew_cost: The price to hire a local crew for t days is as follows.
- total_cost: The sum of the additional crew cost, the local crew cost, and the fine (if applicable) equals the final total cost.
- For every fine value, these computed values are kept in the additional crews per fine and total cost per fine lists.

```
# Convert lists to arrays for easier plotting
total_costs = np.array(total_costs)
crews_needed = np.array(crews_needed)
```

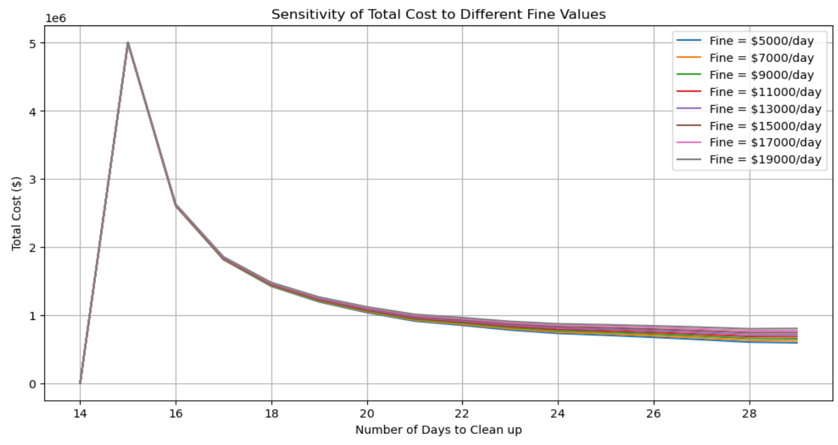
Upon computing the overall expenses and supplementary personnel required for every fine value, the lists are transformed into numpy arrays to facilitate their manipulation and visualization.

```
# Plot the results
plt.figure(figsize=(12, 6))

# Plot total cost vs fine
for idx, fine in enumerate(fine_values):
    plt.plot(cleanup_days, total_costs[idx], label=f'Fine = ${fine}/day')

plt.xlabel('Number of Days to Clean up')
plt.ylabel('Total Cost ($)')
plt.title('Sensitivity of Total Cost to Different Fine Values')
plt.legend()
plt.grid(True)
plt.show()
```

The code generates a plot for various fine values that illustrates how the overall cost varies as the number of cleanup days rises:

- The X-axis shows how many days there are for cleanup.
- The operation's total cost is shown on the Y-axis.
- Each line represents a distinct fine value.

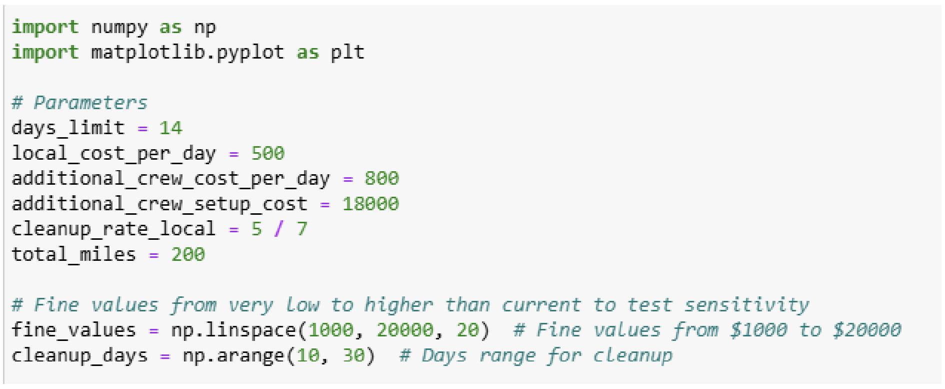
Result



The graph illustrates how the overall cost of the shoreline cleanup operation varies with the number of cleanup days, with varying fine amounts imposed for each extra day beyond the 14-day threshold. Due to the addition of fines and additional crew costs, there is a sudden increase in the total cost on day 15, followed by a gradual decrease as the cleanup gets done. After 16 days, the costs show a gradual convergence across various fine values, suggesting that the variance in fines becomes less significant as the number of cleanup days increases. In general, fines are most sensitive during the first few days (15–18), and costs level off after that.

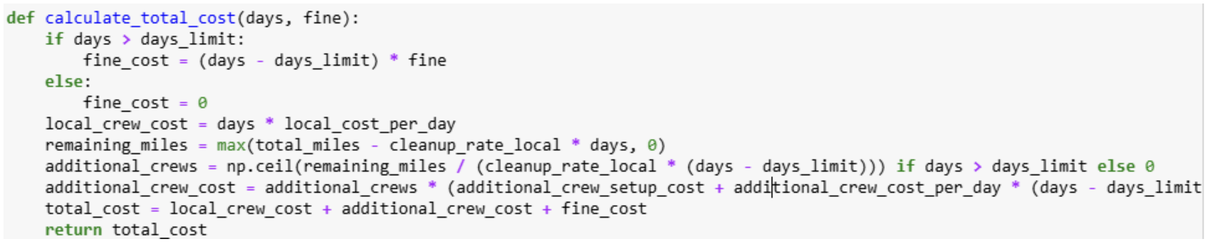
(d) The company has filed an appeal on the grounds that the amount of the fine is excessive. Assuming that the only purpose of the fine is to motivate the company to clean up the oil spill in a timely manner, is the fine excessive?

Explanation of the Code:



```python
import numpy as np
import matplotlib.pyplot as plt

# Parameters
days_limit = 14
local_cost_per_day = 500
additional_crew_cost_per_day = 800
additional_crew_setup_cost = 18000
cleanup_rate_local = 5 / 7
total_miles = 200

# Fine values from very low to higher than current to test sensitivity
fine_values = np.linspace(1000, 20000, 20)  # Fine values from $1000 to $20000
cleanup_days = np.arange(10, 30)  # Days range for cleanup
```

Setting up parameters and creating a range to test different conditions.



```python
def calculate_total_cost(days, fine):
    if days > days_limit:
        fine_cost = (days - days_limit) * fine
    else:
        fine_cost = 0
    local_crew_cost = days * local_cost_per_day
    remaining_miles = max(total_miles - cleanup_rate_local * days, 0)
    additional_crews = np.ceil(remaining_miles / (cleanup_rate_local * (days - days_limit))) if days > days_limit else 0
    additional_crew_cost = additional_crews * (additional_crew_setup_cost + additional_crew_cost_per_day * (days - days_limit
    total_cost = local_crew_cost + additional_crew_cost + fine_cost
    return total_cost
```

This function determines the fine amount as well as the overall cost of the cleanup for a specified number of days.

- fine_cost: There is a fine for each day that the business stays beyond the 14-day limit.
- local_crew_cost: This is just the cost per day for the number of cleanup days that the local crew is used.
- remaining_miles: The number of miles that need to be cleaned after several days of cleaning is determined.
- additional_crew and additional_crew_cost: Should the company need to hire crews for longer than 14 days, there will be additional expenses. The remaining miles are used to determine how many additional crews are needed, and the cost of hiring and maintaining those crews is added.
- total_cost: The sum of the local crew cost, any additional crew costs, and any applicable fines is the total cost.

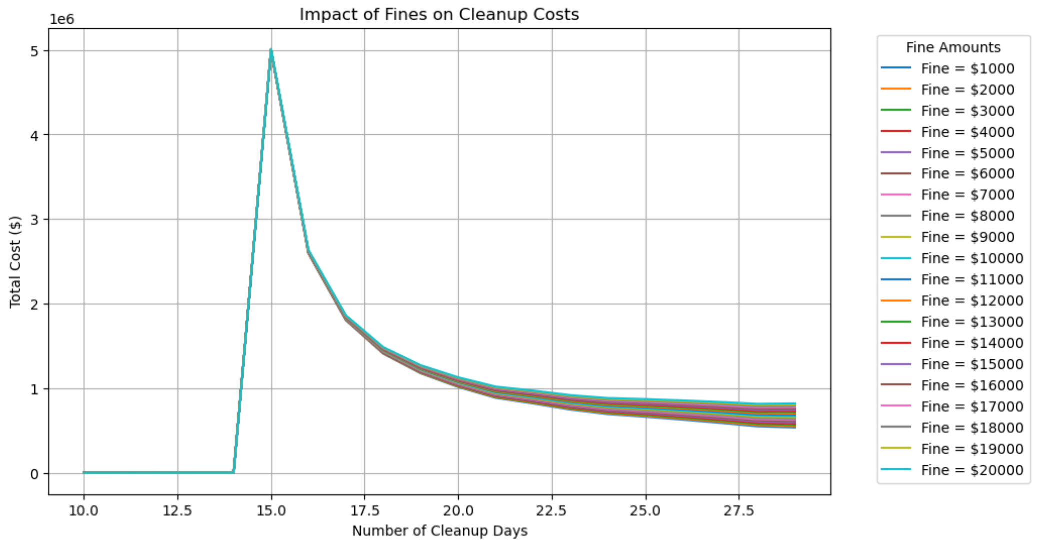
```
# Array to hold results
results = np.zeros((len(fine_values), len(cleanup_days)))

# Compute total costs for each fine and cleanup day combination
for idx, fine in enumerate(fine_values):
    for jdx, day in enumerate(cleanup_days):
        results[idx, jdx] = calculate_total_cost(day, fine)
```

The calculate_total_cost function is then used by the code to determine the overall cost for every combination of fine amount and cleanup days. The outcomes are kept in a 2D. This loop computes the resulting total cost by iterating over each fine value and cleanup day.

```
# Plot results
plt.figure(figsize=(10, 6))
for idx, fine in enumerate(fine_values):
    plt.plot(cleanup_days, results[idx], label=f'Fine = ${fine:.0f}')
plt.xlabel('Number of Cleanup Days')
plt.ylabel('Total Cost ($)')
plt.title('Impact of Fines on Cleanup Costs')
plt.legend(title="Fine Amounts", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
```

- The quantity of cleanup days is shown on the x-axis.
- The entire cost of the cleanup is shown on the y-axis.
- A distinct fine value is represented by each line on the graph.
- The plot makes it easier to see how rising fines impact the overall cost over various cleanup process lengths.

Result

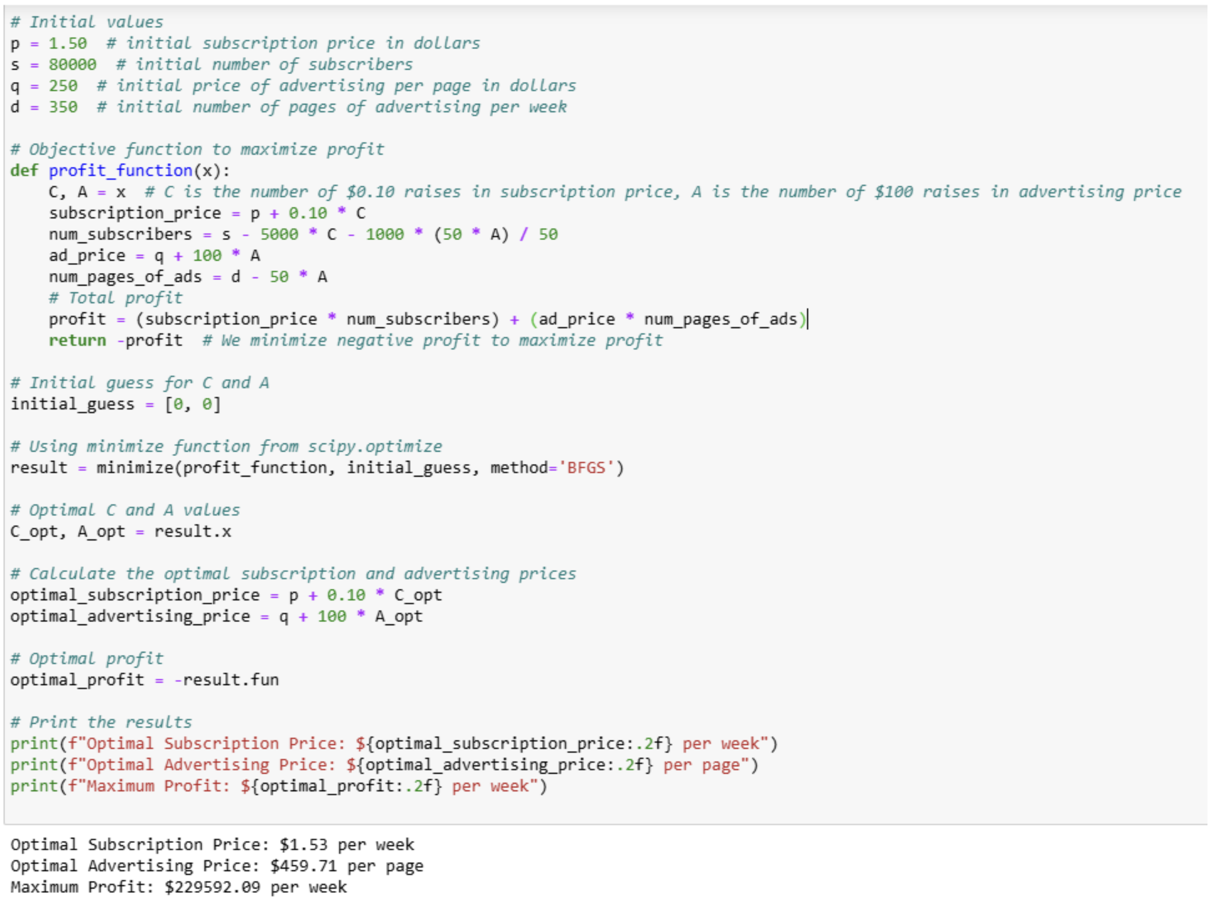


- The graph demonstrates that when the cleanup is finished in less than 14 days—the crucial window of time before fines are imposed—the overall costs stay low.
- There is a noticeable increase in expenses around day 15, suggesting that going over the 14 days results in substantial fines that significantly raise the overall cost, particularly when the fine amounts are higher.
- Day 15 marks the beginning of the end of the total cost as more crews and resources are brought in to finish the cleanup, cutting down on the number of miles left and shortening the duration of penalties.
- After about 20 days, the effects of different fine amounts become less noticeable as all the fine lines converge, indicating that at this point, the operational costs of prolonged cleanup outweigh the impact of fines.

II. A local daily newspaper has recently been acquired by a large media conglomerate. The paper currently sells for \$1.50/week and has a circulation of 80,000 subscribers. Advertising sells for \$250/page, and the paper currently sells 350 pages/week (50 pages/day). The new management is looking for ways to increase profits. It is estimated that an increase of ten cents/week in the subscription price will cause a drop in circulation of 5,000 subscribers. Increasing the price of advertising by \$100/page will cause the paper to lose approximately 50 pages of advertising per week. The loss of advertising will also affect circulation, since one of the reasons people buy the paper is for the advertisements. It is estimated that a loss of 50 pages of advertisements per week will reduce circulation by 1,000 subscriptions.

(a) Find the weekly subscription price and the advertising price that will maximize profit. Use the five-step method, and model as an unconstrained optimization problem.

Python code with result



```python
# Initial values
p = 1.50  # initial subscription price in dollars
s = 80000  # initial number of subscribers
q = 250  # initial price of advertising per page in dollars
d = 350  # initial number of pages of advertising per week

# Objective function to maximize profit
def profit_function(x):
    C, A = x  # C is the number of $0.10 raises in subscription price, A is the number of $100 raises in advertising price
    subscription_price = p + 0.10 * C
    num_subscribers = s - 5000 * C - 1000 * (50 * A) / 50
    ad_price = q + 100 * A
    num_pages_of_ads = d - 50 * A
    # Total profit
    profit = (subscription_price * num_subscribers) + (ad_price * num_pages_of_ads)
    return -profit  # We minimize negative profit to maximize profit

# Initial guess for C and A
initial_guess = [0, 0]

# Using minimize function from scipy.optimize
result = minimize(profit_function, initial_guess, method='BFGS')

# Optimal C and A values
C_opt, A_opt = result.x

# Calculate the optimal subscription and advertising prices
optimal_subscription_price = p + 0.10 * C_opt
optimal_advertising_price = q + 100 * A_opt

# Optimal profit
optimal_profit = -result.fun

# Print the results
print(f"Optimal Subscription Price: ${optimal_subscription_price:.2f} per week")
print(f"Optimal Advertising Price: ${optimal_advertising_price:.2f} per page")
print(f"Maximum Profit: ${optimal_profit:.2f} per week")
```

```
Optimal Subscription Price: $1.53 per week
Optimal Advertising Price: $459.71 per page
Maximum Profit: $229592.09 per week
```

Step 1

Variables:

p = Price of paper subscription

s= Number of subscribers

q= Price of advertising

d= Number of pagers of adds

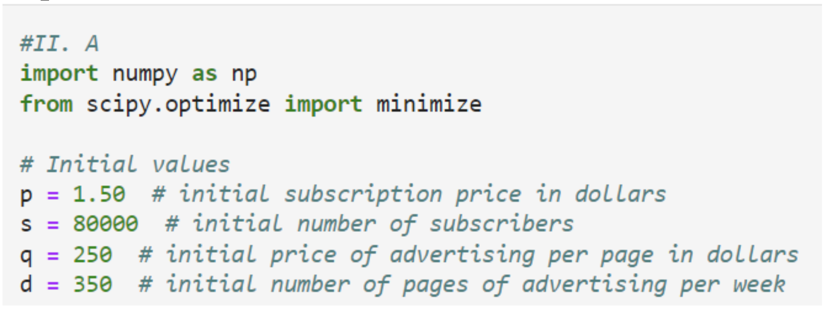
C= Number of \$0.10 raises in subscribers price

A= Number of \$100 raises in subscribers price

P= Profit

Assumptions:
s = $80{,}000 - 5{,}000C - 1{,}000A$
d = $350 - 50A$
p = $1.50 + 0.10C$
q = $250 + 100A$

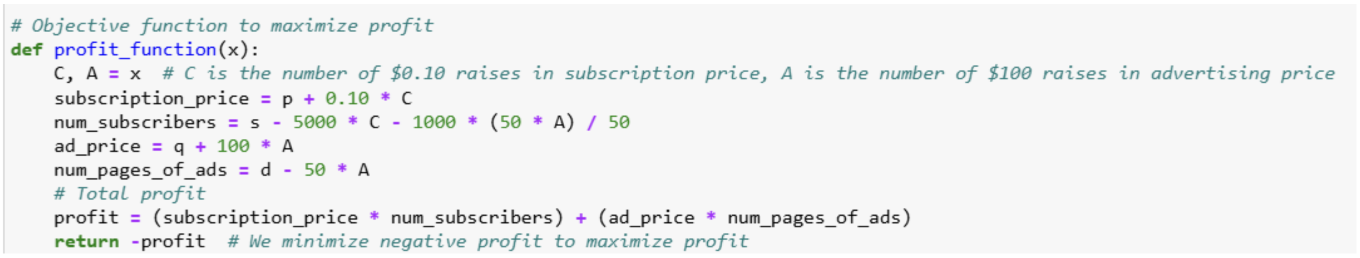
Explanation of Code



```python
#II. A
import numpy as np
from scipy.optimize import minimize

# Initial values
p = 1.50  # initial subscription price in dollars
s = 80000  # initial number of subscribers
q = 250  # initial price of advertising per page in dollars
d = 350  # initial number of pages of advertising per week
```

The program uses numpy (as np) to perform numerical operations.

The minimize function in scipy. optimize is used to determine the variables that minimize (or maximize) a given function.



```python
# Objective function to maximize profit
def profit_function(x):
    C, A = x  # C is the number of $0.10 raises in subscription price, A is the number of $100 raises in advertising price
    subscription_price = p + 0.10 * C
    num_subscribers = s - 5000 * C - 1000 * (50 * A) / 50
    ad_price = q + 100 * A
    num_pages_of_ads = d - 50 * A
    # Total profit
    profit = (subscription_price * num_subscribers) + (ad_price * num_pages_of_ads)
    return -profit  # We minimize negative profit to maximize profit
```

The profit_function describes how profit is calculated using two variables:

C: The subscription price increases by $0.10.
A: The number of $100 increases in advertising prices.

Calculation for New Subscription Price:

$$subscription_{\_price} = p + 0.10\ (C)$$

This depicts how the price rises by $0.10 (C) increments.

Calculation for New Number of Subscribers:

$$num_{subscribers} = s - 5000\ (C) - 1000\ \left(\frac{50\ (A)}{50}\right)$$

This shows the way the total amount of subscribers reduces by 5,000 for each $0.10 increase in price (C), and by 1,000 subscribers for every 50 pages of ads lost because of increased advertising prices.

Calculation for New Advertising Price:

$$ad_{price} = q + 100\ (A)$$

Represents how the advertising price increases by $100 for every increment of (A).

Calculation for New Number of Pages of Ads:

$$num_{pages_{of\,ads}} = d - 50\,(A)$$

This shows how the total amount of ad pages reduces by 50% for every $100 rise in the ad price.

The total profit is the sum of:

- Subscription Revenue:
$$subscription_{price}(num_{subscribers})$$
- Advertising Revenue:
$$ad\_price\,(num_{pages_{of\,ads}})$$

Because minimize () seeks the lowest possible value, the function returns the negative of the profit. By reducing the negative profit, the function maximizes the actual profit.

```
# Initial guess for C and A
initial_guess = [0, 0]
```

- This is an initial estimate for the variables C (number of $0.10 subscription price increases) and A (number of $100 advertising price increases).
- We begin with the premise that there will be no price increases (C = 0, A = 0).

```
# Using minimize function from scipy.optimize
result = minimize(profit_function, initial_guess, method='BFGS')
```

- The minimize () function is used to determine the optimal values of C and A for profit.
- 'BFGS' is a popular optimization algorithm for minimizing the profit function.
- It returns an object containing information about the optimization, such as the optimal values for C and A.

```
# Optimal C and A values
C_opt, A_opt = result.x
```

- result.x contains the best possible values for C (the number of $0.10 a subscription price increases) and A (the number of $100 advertising price rises).
- These values are kept in C_opt and A_opt.

```
# Calculate the optimal subscription and advertising prices
optimal_subscription_price = p + 0.10 * C_opt
optimal_advertising_price = q + 100 * A_opt
```

- The optimal subscription price is obtained through adding a maximum of $0.10 increases (C_opt) into the initial subscription price (p).
- The optimal advertising price can be determined by adding the optimal number of $100 raises (A_opt) to the first advertising price (q).

```
# Optimal profit
optimal_profit = -result.fun
```

- result.fun returns the value of the minimized function (that is, the negative of the maximum profit).
- By negating it (-result.fun), we get exactly the maximum profit.

```python
# Print the results
print(f"Optimal Subscription Price: ${optimal_subscription_price:.2f} per week")
print(f"Optimal Advertising Price: ${optimal_advertising_price:.2f} per page")
print(f"Maximum Profit: ${optimal_profit:.2f} per week")
```

```
Optimal Subscription Price: $1.53 per week
Optimal Advertising Price: $459.71 per page
Maximum Profit: $229592.09 per week
```

- By printing the result, we get the optimal subscription price of $1.53 per week
- Optimal Advertising Price for $459.71 per page
- Maximum Profit for $229,592.09 per week

(b) Examine the sensitivity of your conclusions in part (a) to the assumption of 5,000 lost sales when the price of the paper increases by ten cents.

Python Code

```python
#II. B
import numpy as np
from scipy.optimize import minimize

# Initial values
p = 1.50  # initial subscription price in dollars
s = 80000  # initial number of subscribers
q = 250  # initial price of advertising per page in dollars
d = 350  # initial number of pages of advertising per week
loss_values = [4000, 5000, 6000, 7000]  # Different assumptions for lost subscribers per $0.10 increase

# Function to compute profit based on variable loss rate
def profit_function(x, loss_rate):
    C, A = x  # C is the number of $0.10 raises in subscription price, A is the number of $100 raises in advertising price
    subscription_price = p + 0.10 * C
    num_subscribers = s - loss_rate * C - 1000 * (50 * A) / 50
    ad_price = q + 100 * A
    num_pages_of_ads = d - 50 * A
    # Total profit
    profit = (subscription_price * num_subscribers) + (ad_price * num_pages_of_ads)
    return -profit  # We minimize negative profit to maximize profit

# Loop through different lost subscribers per $0.10 increase and compute results
for loss_rate in loss_values:
    # Define a new objective function based on the current loss rate
    result = minimize(lambda x: profit_function(x, loss_rate), [0, 0], method='BFGS')

    # Optimal C and A values
    C_opt, A_opt = result.x

    # Calculate the optimal subscription and advertising prices
    optimal_subscription_price = p + 0.10 * C_opt
    optimal_advertising_price = q + 100 * A_opt

    # Optimal profit
    optimal_profit = -result.fun

    # Print the results for the current loss rate
    print(f"Loss of {loss_rate} subscribers per $0.10 increase:")
    print(f"  Optimal Subscription Price: ${optimal_subscription_price:.2f} per week")
    print(f"  Optimal Advertising Price: ${optimal_advertising_price:.2f} per page")
    print(f"  Maximum Profit: ${optimal_profit:.2f} per week")
    print("--------------------------------------------------")
```

Explanation of Code:
Additional code from (a)

```
loss_values = [4000, 5000, 6000, 7000]  # Different assumptions for lost subscribers per $0.10 increase
```

- loss_values is a list of different assumptions about how many subscribers are lost for every $0.10 increase in subscription price. The values are 4,000, 5,000, 6,000, and 7,000.

```
# Function to compute profit based on variable loss rate
def profit_function(x, loss_rate):
    C, A = x  # C is the number of $0.10 raises in subscription price, A is the number of $100 raises in advertising price
    subscription_price = p + 0.10 * C
    num_subscribers = s - loss_rate * C - 1000 * (50 * A) / 50
```

Code for New Number Subscribers:

$$num_{subscribers} = s - loss_{rate}(C - 1000)\left(\frac{50\ (A)}{50}\right)$$

- A decrease in loss_rate subscribers per $0.10 price increase (C).
- A 1,000-subscriber loss for every 50 pages of ads is lost as advertising prices rise (A).

```
# Loop through different lost subscribers per $0.10 increase and compute results
for loss_rate in loss_values:
    # Define a new objective function based on the current loss rate
    result = minimize(lambda x: profit_function(x, loss_rate), [0, 0], method='BFGS')
```

- The program iterates using various loss_values (4,000, 5,000, 6,000, and 7,000) to calculate the subscribers lost per $0.10 increase.
- For each loss_rate, the minimize () function is used to determine the optimal values of C (subscription price increase) and A (advertising price increase) to maximize profit.

```
result = minimize(lambda x: profit_function(x, loss_rate), [0, 0], method='BFGS')
```

- The amounts of C and A that lessen the profit function for the current loss_rate is determined by the minimize () function using the BFGS algorithm.
- For C and A, the initial guess is [0, 0]; that is, there are no price increases at first.

Result:

```
Loss of 4000 subscribers per $0.10 increase:
  Optimal Subscription Price: $1.72 per week
  Optimal Advertising Price: $457.76 per page
  Maximum Profit: $231555.07 per week
--------------------------------------------------
Loss of 5000 subscribers per $0.10 increase:
  Optimal Subscription Price: $1.53 per week
  Optimal Advertising Price: $459.71 per page
  Maximum Profit: $229592.09 per week
--------------------------------------------------
Loss of 6000 subscribers per $0.10 increase:
  Optimal Subscription Price: $1.40 per week
  Optimal Advertising Price: $461.01 per page
  Maximum Profit: $230160.55 per week
--------------------------------------------------
Loss of 7000 subscribers per $0.10 increase:
  Optimal Subscription Price: $1.31 per week
  Optimal Advertising Price: $461.94 per page
  Maximum Profit: $232174.77 per week
--------------------------------------------------
```

- As the number of lost subscribers rises (from 4,000 to 7,000), the optimal subscription price falls because a higher loss prevents large price increases.
- The optimal advertising price rises a little as the loss rate rises, perhaps to offset the decrease in subscription revenue.
- The maximum profit fluctuates but generally remains within a range, indicating that the model is relatively resistant to changes in the assumed subscriber loss rate.

(c) Examine the sensitivity of your conclusions in part (a) to the assumption of 50 pages/week of lost advertising sales when the price of advertising is increased by \$100/page.

Python Code

```python
import numpy as np
from scipy.optimize import minimize

# Initial values
p = 1.50  # initial subscription price in dollars
s = 80000  # initial number of subscribers
q = 250  # initial price of advertising per page in dollars
d = 350  # initial number of pages of advertising per week
loss_pages_values = [40, 50, 60, 70]  # Different assumptions for lost ad pages per $100 increase

# Function to compute profit based on variable lost ad pages
def profit_function(x, lost_pages):
    C, A = x  # C is the number of $0.10 raises in subscription price, A is the number of $100 raises in advertising price
    subscription_price = p + 0.10 * C
    num_subscribers = s - 5000 * C - 1000 * (lost_pages * A) / 50
    ad_price = q + 100 * A
    num_pages_of_ads = d - lost_pages * A
    # Total profit
    profit = (subscription_price * num_subscribers) + (ad_price * num_pages_of_ads)
    return -profit  # We minimize negative profit to maximize profit

# Loop through different lost ad pages per $100 increase and compute results
for lost_pages in loss_pages_values:
    # Define a new objective function based on the current lost pages assumption
    result = minimize(lambda x: profit_function(x, lost_pages), [0, 0], method='BFGS')

    # Optimal C and A values
    C_opt, A_opt = result.x

    # Calculate the optimal subscription and advertising prices
    optimal_subscription_price = p + 0.10 * C_opt
    optimal_advertising_price = q + 100 * A_opt

    # Optimal profit
    optimal_profit = -result.fun

    # Print the results for the current lost pages assumption
    print(f"Loss of {lost_pages} ad pages per $100 increase:")
    print(f"  Optimal Subscription Price: ${optimal_subscription_price:.2f} per week")
    print(f"  Optimal Advertising Price: ${optimal_advertising_price:.2f} per page")
    print(f"  Maximum Profit: ${optimal_profit:.2f} per week")
    print("--------------------------------------------------")
```

Explanation of Code:
Additional code from (b)

```python
loss_pages_values = [40, 50, 60, 70]  # Different assumptions for lost ad pages per $100 increase
```

- loss_pages_values: A set of assumptions about the amount of ad pages lost for every \$100 raised in the advertising price. The numbers are 40, 50, 60, and 70 ad pages.

```python
# Function to compute profit based on variable lost ad pages
def profit_function(x, lost_pages):
    C, A = x  # C is the number of $0.10 raises in subscription price, A is the number of $100 raises in advertising price
    subscription_price = p + 0.10 * C
    num_subscribers = s - 5000 * C - 1000 * (lost_pages * A) / 50
    ad_price = q + 100 * A
    num_pages_of_ads = d - lost_pages * A
    # Total profit
    profit = (subscription_price * num_subscribers) + (ad_price * num_pages_of_ads)
    return -profit  # We minimize negative profit to maximize profit
```

Code for New Number Subscribers:

$$num_{subscribers} = s - 5000\ (C - 1000)\ \left(\frac{lost\_pages\ (A)}{50}\right)$$

The number of subscribers declines due to two factors:

- Each $0.10 price increase results in a loss of 5,000 subscribers (C).
- A 1,000-subscriber loss for every 50 pages of ads is lost as advertising prices rise. The value of lost_pages vary depending on the current assumption.

```python
# Loop through different lost ad pages per $100 increase and compute results
for lost_pages in loss_pages_values:
    # Define a new objective function based on the current lost pages assumption
    result = minimize(lambda x: profit_function(x, lost_pages), [0, 0], method='BFGS')
```

- The list loss_pages_values, which has various estimates for the number of ad pages lost for every $100 rise in the advertising price, is looped through by the program. 40, 50, 60, and 70 pages are the values.
- The minimize () function is used to determine the values of C (subscription price increase) and A (advertising price increase) for each value of lost_pages to achieve the greatest profit under the current assumption.

Result

```
Loss of 40 ad pages per $100 increase:
  Optimal Subscription Price: $1.53 per week
  Optimal Advertising Price: $547.24 per page
  Maximum Profit: $242936.85 per week
--------------------------------------------------
Loss of 50 ad pages per $100 increase:
  Optimal Subscription Price: $1.53 per week
  Optimal Advertising Price: $459.71 per page
  Maximum Profit: $229592.09 per week
--------------------------------------------------
Loss of 60 ad pages per $100 increase:
  Optimal Subscription Price: $1.53 per week
  Optimal Advertising Price: $401.35 per page
  Maximum Profit: $221352.29 per week
--------------------------------------------------
Loss of 70 ad pages per $100 increase:
  Optimal Subscription Price: $1.53 per week
  Optimal Advertising Price: $359.65 per page
  Maximum Profit: $216029.94 per week
--------------------------------------------------
```

The code examines the effects of various assumptions on the quantity of lost subscribers per $0.10 increase in price on the best price for advertising, subscriptions, and maximum profit.

- The ideal subscription price drops as the number of lost customers rises (from 4,000 to 7,000) because a larger loss deters significant price increases.
- It is possible that the optimal advertising price rises slightly in response to an increase in loss rate, perhaps to offset a decline in subscription revenue.
- The maximum profit varies but mostly remains within a range, indicating a shift in the assumed subscriber loss rate does not significantly affect the model.