COMP 9057 - DECISION ANALYTICS

ASSIGNMENT 2 - Linear Programming (Report)

Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

TASK 1 (supply chain, 21 points)

A. Solution

- Load the spreadsheet and import the data.
  I have used openpyxl library in python and defined load_sheet function (from line 6 to line 20) to load the sheets to supplier_stock, raw_mat_costs, raw_mat_ship, prod_req, prod_cap, prod_cost, customer_demand and ship_cost variables (from line 53 to 68).
- And I created decision variables for the orders from the suppliers, production volume and delivery to customers as supplier_order_vars, production_vol_vars, customer_delivery_vars.

```
    # 2. Define the variables
    # supplier order variables
    supplier_order_vars = {}
    for supplier in suppliers:
        for material in materials:
            for factory in factories:
                supplier_order_vars[supplier, material, factory] = solver.IntVar(0, 
solver.infinity(), f'supplier_order[{supplier}, {material}, {factory}]')

    # production volume variables
    production_vol_vars = {}
    for product in products:
        for factory in factories:
            production_vol_vars[product, factory] = solver.IntVar(0, solver.infinity(), 
f'production_volume[{product}, {factory}]')

    # customer delivery variables
    customer_delivery_vars = {}
    for customer in customers:
        for product in products:
            for factory in factories:
                customer_delivery_vars[customer, product, factory] =\
                    solver.IntVar(0, solver.infinity(), f'customer_delivery[{customer}, 
{product}, {factory}]')
```

- Then, I implemented each constraint using solver.Add function.
  - Ensure factories produce more than they ship to the customers
    For all products and factories, the production volume var must be greater or equal than the sum of customer delivery amount for all customers.

```
for product in products:
        for factory in factories:
            solver.Add(production_vol_vars[product, factory] - 
sum([customer_delivery_vars[customer, product, factory] for customer in 
customers]) >= 0)
```

  - Ensure that customer demand is met
    For all customer and product, the sum of customer delivery amount for all factories must greater or equal than the customer demand.

```
for customer in customers:
    for product in products:
        solver.Add(sum([customer_delivery_vars[customer, product, factory] for
factory in factories]) >= customer_demand[product, customer])
```

- Ensure that suppliers have all ordered items in stock

```
 # 5. Define and implement the constraints that ensure that suppliers have all
ordered items in stock
    for supplier in suppliers:
        for material in materials:
            solver.Add(sum([supplier_order_vars[supplier, material, factory] for
factory in factories]) <= supplier_stock[supplier, material])
```

```
for material in materials:
        for factory in factories:
            solver.Add(sum([supplier_order_vars[supplier, material, factory] for
supplier in suppliers]) -\
                       sum([production_vol_vars[product, factory] * prod_req[product,
material] for product in products]) >= 0)
```

- Ensure that the manufacturing capacities are not exceeded
  For all products and factories, the production volume must be less or equal than product capacity.

```
 # 6. Define and implement the constraints that ensure that the manufacturing
capacities are not exceeded
    for product in products:
        for factory in factories:
            solver.Add(production_vol_vars[product, factory] <= prod_cap[product,
factory])
```

- Then, finally I have defined the objective function.

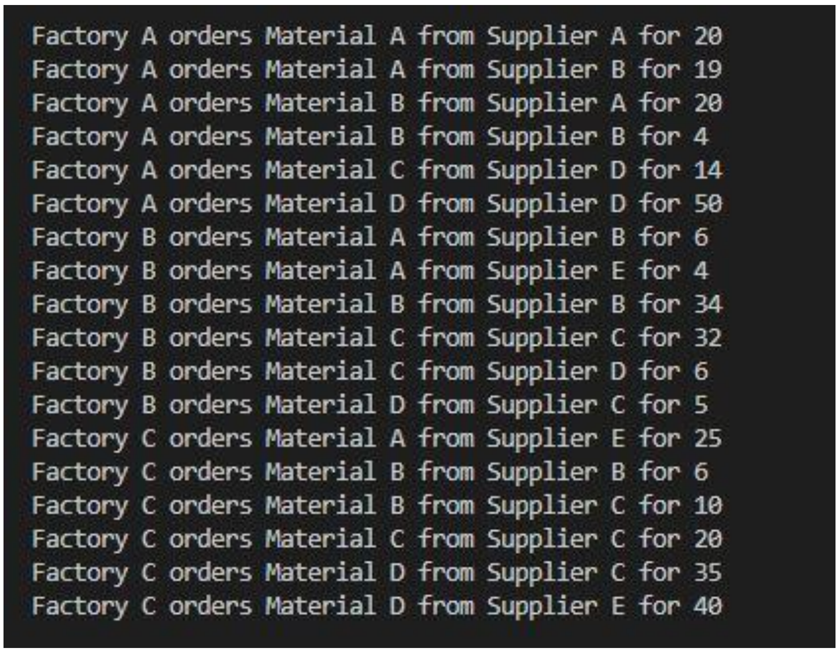
  ```
  overall_cost = supplier_cost + supplier_ship_cost + production_cost + customer_ship_cost

  solver.Minimize(overall_cost)
  ```

  Here, I defined supplier_cost, supplier_ship_cost, production_cost, customer_ship_cost separately(from line 124 to line 135).
- Then solved the program using solver.Solve (line 141).

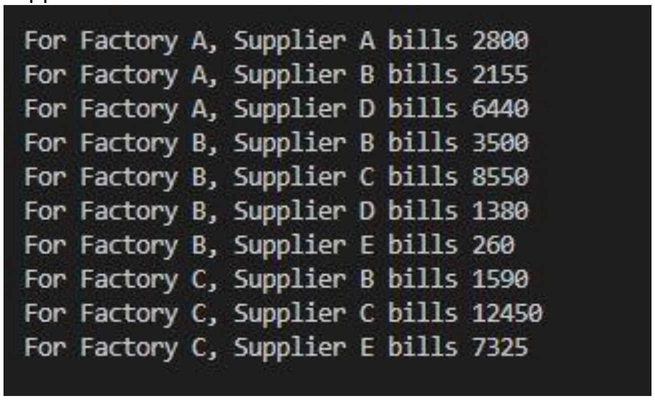
B. Result

I printed each required results on the optimal solution.
The overall cost was: 49315

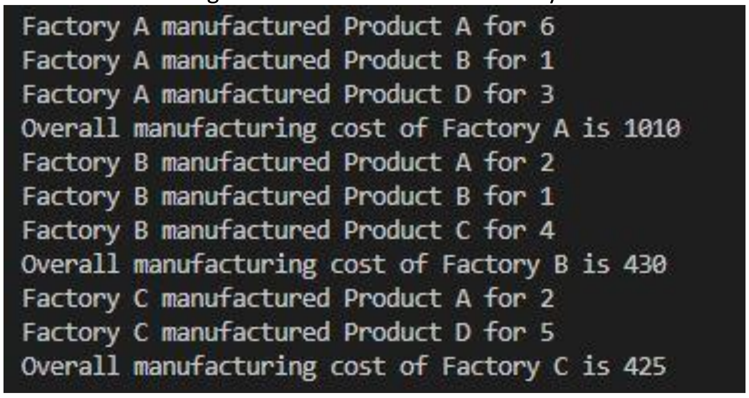
- Determine for each factory how much material has to be ordered from each individual supplier.


```
Factory A orders Material A from Supplier A for 20
Factory A orders Material A from Supplier B for 19
Factory A orders Material B from Supplier A for 20
Factory A orders Material B from Supplier B for 4
Factory A orders Material C from Supplier D for 14
Factory A orders Material D from Supplier D for 50
Factory B orders Material A from Supplier B for 6
Factory B orders Material A from Supplier E for 4
Factory B orders Material B from Supplier B for 34
Factory B orders Material C from Supplier C for 32
Factory B orders Material C from Supplier D for 6
Factory B orders Material D from Supplier C for 5
Factory C orders Material A from Supplier E for 25
Factory C orders Material B from Supplier B for 6
Factory C orders Material B from Supplier C for 10
Factory C orders Material C from Supplier C for 20
Factory C orders Material D from Supplier C for 35
Factory C orders Material D from Supplier E for 40
```

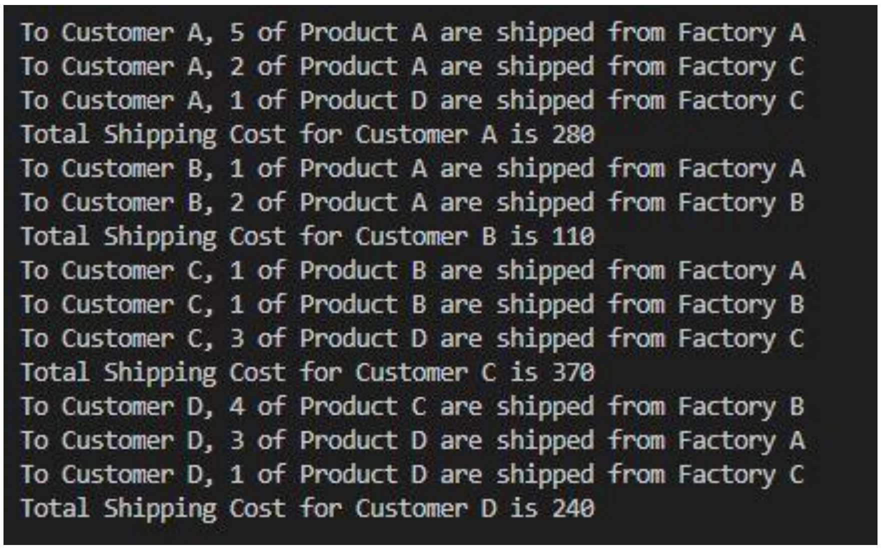
- Determine for each factory what the supplier bill comprising material cost and delivery will be for each supplier.


```
For Factory A, Supplier A bills 2800
For Factory A, Supplier B bills 2155
For Factory A, Supplier D bills 6440
For Factory B, Supplier B bills 3500
For Factory B, Supplier C bills 8550
For Factory B, Supplier D bills 1380
For Factory B, Supplier E bills 260
For Factory C, Supplier B bills 1590
For Factory C, Supplier C bills 12450
For Factory C, Supplier E bills 7325
```

- Determine for each factory how many units of each product are being manufactured. Also determine the total manufacturing cost for each individual factory.

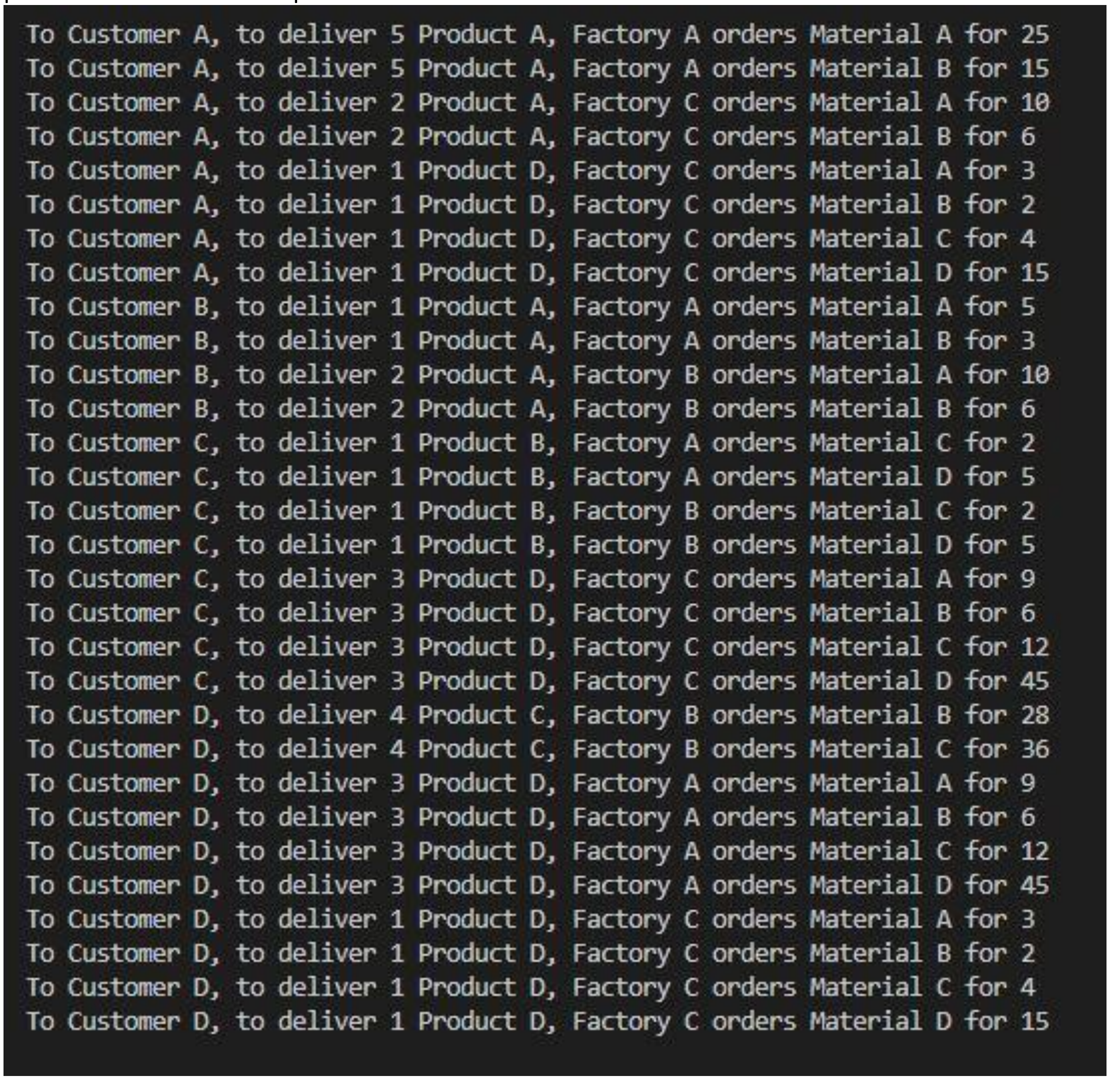

```
Factory A manufactured Product A for 6
Factory A manufactured Product B for 1
Factory A manufactured Product D for 3
Overall manufacturing cost of Factory A is 1010
Factory B manufactured Product A for 2
Factory B manufactured Product B for 1
Factory B manufactured Product C for 4
Overall manufacturing cost of Factory B is 430
Factory C manufactured Product A for 2
Factory C manufactured Product D for 5
Overall manufacturing cost of Factory C is 425
```

- Determine for each customer how many units of each product are being shipped from each factory. Also determine the total shipping cost per customer.



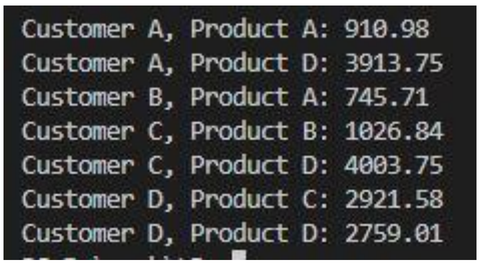
```
To Customer A, 5 of Product A are shipped from Factory A
To Customer A, 2 of Product A are shipped from Factory C
To Customer A, 1 of Product D are shipped from Factory C
Total Shipping Cost for Customer A is 280
To Customer B, 1 of Product A are shipped from Factory A
To Customer B, 2 of Product A are shipped from Factory B
Total Shipping Cost for Customer B is 110
To Customer C, 1 of Product B are shipped from Factory A
To Customer C, 1 of Product B are shipped from Factory B
To Customer C, 3 of Product D are shipped from Factory C
Total Shipping Cost for Customer C is 370
To Customer D, 4 of Product C are shipped from Factory B
To Customer D, 3 of Product D are shipped from Factory A
To Customer D, 1 of Product D are shipped from Factory C
Total Shipping Cost for Customer D is 240
```

- Determine for each customer the fraction of each material each factory has to order for manufacturing products delivered to that particular customer.



```
To Customer A, to deliver 5 Product A, Factory A orders Material A for 25
To Customer A, to deliver 5 Product A, Factory A orders Material B for 15
To Customer A, to deliver 2 Product A, Factory C orders Material A for 10
To Customer A, to deliver 2 Product A, Factory C orders Material B for 6
To Customer A, to deliver 1 Product D, Factory C orders Material A for 3
To Customer A, to deliver 1 Product D, Factory C orders Material B for 2
To Customer A, to deliver 1 Product D, Factory C orders Material C for 4
To Customer A, to deliver 1 Product D, Factory C orders Material D for 15
To Customer B, to deliver 1 Product A, Factory A orders Material A for 5
To Customer B, to deliver 1 Product A, Factory A orders Material B for 3
To Customer B, to deliver 2 Product A, Factory B orders Material A for 10
To Customer B, to deliver 2 Product A, Factory B orders Material B for 6
To Customer C, to deliver 1 Product B, Factory A orders Material C for 2
To Customer C, to deliver 1 Product B, Factory A orders Material D for 5
To Customer C, to deliver 1 Product B, Factory B orders Material C for 2
To Customer C, to deliver 1 Product B, Factory B orders Material D for 5
To Customer C, to deliver 3 Product D, Factory C orders Material A for 9
To Customer C, to deliver 3 Product D, Factory C orders Material B for 6
To Customer C, to deliver 3 Product D, Factory C orders Material C for 12
To Customer C, to deliver 3 Product D, Factory C orders Material D for 45
To Customer D, to deliver 4 Product C, Factory B orders Material B for 28
To Customer D, to deliver 4 Product C, Factory B orders Material C for 36
To Customer D, to deliver 3 Product D, Factory A orders Material A for 9
To Customer D, to deliver 3 Product D, Factory A orders Material B for 6
To Customer D, to deliver 3 Product D, Factory A orders Material C for 12
To Customer D, to deliver 3 Product D, Factory A orders Material D for 45
To Customer D, to deliver 1 Product D, Factory C orders Material A for 3
To Customer D, to deliver 1 Product D, Factory C orders Material B for 2
To Customer D, to deliver 1 Product D, Factory C orders Material C for 4
To Customer D, to deliver 1 Product D, Factory C orders Material D for 15
```

- Based on this calculate the overall unit cost of each product per customer including the raw materials used for the manufacturing of the customer's specific product, the cost of manufacturing for the specific customer and all relevant shipping costs.



```
Customer A, Product A: 910.98
Customer A, Product D: 3913.75
Customer B, Product A: 745.71
Customer C, Product B: 1026.84
Customer C, Product D: 4003.75
Customer D, Product C: 2921.58
Customer D, Product D: 2759.01
```

TASK 2 (delivery driver, 8 points)

A. Solution

I used OR Tools' CBC_MIXED_INTEGER_PROGRAMMING solver to solve this problem.

- I loaded the xlsx spread sheet using load_sheet function defined previously to defined distance var. (line 236 – 242)
- Defined the decision variable with 0 or 1 integer var for all pairs of towns that are not the equal.

```
# 2. For each pair of towns that need to be visited create a decision variable to decide
if this leg should be included into the route
    legs = {}

    for town1 in towns_to_visit:
        for town2 in towns_to_visit:
            if town1 != town2:
                legs[town1, town2] = solver.IntVar(0, 1, "")
```

- And defined each constraint.
  - Ensure that the delivery driver arrives in each of the towns that need to be visited.
    For all towns, the sum of arrival leg count must be exactly 1.

```
# 3. Define and implement the constraints that ensure that the delivery driver
arrives in each of the towns that need to be visited
        solver.Add(sum(legs[town, town2] for town2 in towns_to_visit if town2 !=
town) == 1)
```

  - Ensure that the driver departs each of the towns that need to be visited.
    The same as previous, though the sum of departure leg count must be exactly 1.

```
# 4. Define and implement the constraints that ensure that the driver departs each of
the towns that need to be visited
        solver.Add(sum(legs[town1, town] for town1 in towns_to_visit if town !=
town1) == 1)
```

  - Ensure that there are no disconnected self-contained circles in the route.
    For all subtown collections, I defined that the count of all legs must less than the subtown count.

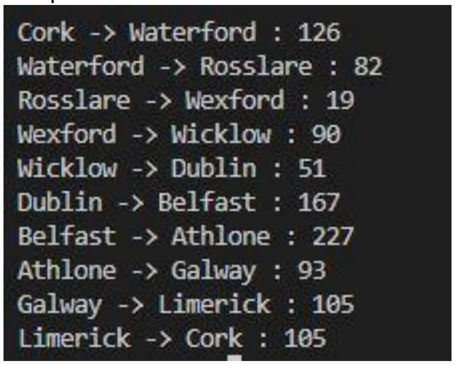
```
# 5. Define and implement the constraints that ensure that there are no disconnected
self-contained circles in the route
    subtowns = [subtown for i in range(2, len(towns_to_visit)) for subtown in
combinations(towns_to_visit, i)]

    for subtown in subtowns:
        solver.Add(sum(legs[town1, town2] for town1 in subtown for town2 in subtown
if town1 != town2) <= len(subtown) - 1)
```

- And defined the objective function to be overall distance must be minimized.

```
# 6. Define and implement the objective function to minimise the overall distance
travelled.
    overall_distance = sum(legs[town1, town2] * distance[town1, town2] for town1 in
towns_to_visit for town2 in towns_to_visit if town1 != town2)
    solver.Minimize(overall_distance)
```

B. Result

The result was as below and the shortest distance was 1065
The path was:



```
Cork -> Waterford : 126
Waterford -> Rosslare : 82
Rosslare -> Wexford : 19
Wexford -> Wicklow : 90
Wicklow -> Dublin : 51
Dublin -> Belfast : 167
Belfast -> Athlone : 227
Athlone -> Galway : 93
Galway -> Limerick : 105
Limerick -> Cork : 105
```

TASK 3 (investment portfolio, 21 points)

A. Solution

- I load the xlsx spreadsheet using load_sheet_3 function (defined from line 21 to 36), then convert it USD or EUR consistently, calculate the monthly return, and overall monthly average reward for single position.
    - Load the spreadsheet (from line 289 to 302). To variables usd_data, eur_data, currency_data, timestamps, stocks
    - Convert the currency (from line 308 to 313).

```
if currency == 'USD':
        for key, value in eur_data.items():
            eur_data[key] = currency_data[key[0], 'EURUSD'] * value
    else:
        for key, value in usd_data.items():
            usd_data[key] = value / currency_data[key[0], 'EURUSD']
```

    - Calculate the return data for each month (except the first month).

```
# calculate the monthly return
    return_data = {}
    for i in range(1, len(timestamps)):
        for stock in stocks:
            return_data[timestamps[i], stock] = \
                stocks_data[timestamps[i], stock] /stocks_data[timestamps[i - 1], 
stock]

    print(f"task3_A_{currency}")
```

    - And calculate the overall monthly average reward for each single position.

```
for stock in stocks:
        average_reward_data[stock] = sum([return_data[timestamp, stock] for 
timestamp in timestamps[1:]]) / (len(timestamps) - 1)
        print(f"The overall average monthly reward of {stock} is 
{average_reward_data[stock]}")
```

- I dig into Solution B part to determine the optimal marketing time. I defined the GLOP solver.
    - Defined the decision variable that indicates the percentage between 0 to 1 for all positions. (positions would be stocks + 'Cash').

```
# For each month create decision variables that indicate the percentage of each 
position held as well as
    # the percentage of cash not invested during this month
    percent_var = {}
    positions = stocks + ['Cash']
    for timestamp in timestamps:
        for position in positions:
            percent_var[timestamp, position] = solver1.NumVar(0, 1, "")
```

    - And defined the constraint to ensure that the investment portfolio always adds up to 100%.

```
# Identify and create the implicit constraints to ensure that the investment
portfolio always adds up to 100%
    for timestamp in timestamps:
        solver1.Add(sum(percent_var[timestamp, position] for position in positions)
== 1.0)
```

- Defined the constraint to make each position not exceed 30% so that in this case would be 0.3.
  So for each timestamp and position, the percentage var must be equal or less than 0.3.

```
# Investing everything into one single position is not good practice.
# Therefore, identify and create constraints that ensure that no single investment position is
ever more than 30% of the overall portfolio
    for timestamp in timestamps:
        for position in positions:
            solver1.Add(percent_var[timestamp, position] <= 0.3)
```

- Finally, I defined the objective function to maximize the summing up all the monthly returns.
  For that, I first for each timestamp, implemented weighed sum with the percentage var for the return values calculated for each stock. (For cash, there would be no return)

```
return_vars = {}
    for timestamp in timestamps[1:]:
        return_vars[timestamp] = sum(percent_var[timestamp, stock] * return_data[timestamp,
stock] for stock in stocks)
```

  Then, define the objective function to maximize the sum of return variables for all timestamps.

```
solver1.Maximize(sum(return_vars[timestamp] for timestamp in timestamps[1:]))
```

- I solved the C part using GLOP solver to minimize the risk.
  - Created the decision variable for percentage of each stock(no cash) on portfolio_vars which are Numerical Variables between 0 to 1.

```
# Create decision variables that indicate the percentage of each position held in
the portfolio during the entire investment period
    portfolio_vars = {}
    for timestamp in timestamps:
        for stock in stocks:
            portfolio_vars[timestamp, stock] = solver2.NumVar(0, 1, "")
```

  - Then, defined the constraint that the investment portfolio always adds up to 100%.
    For all timestamps, sum of portfolio vars of all stocks must be exactly 1.

```
# Create the implicit constraint that the investment portfolio always adds up to
100%
    for timestamp in timestamps:
        solver2.Add(sum(portfolio_vars[timestamp, stock] for stock in stocks) ==
1.0)
```

  - And similarly, defined the constraint that all position percentage must not exceed 30%.

```
# Identify and create constraints to ensure that no single investment
position is ever more than 30% of the overall portfolio
    for timestamp in timestamps:
        for stock in stocks:
            solver2.Add(portfolio_vars[timestamp, stock] <= 0.3)
```

  - And the constraint that the overall average monthly reward must exceed 0.5% so that in that case this would be 1.005.

```
# Create a constraint to ensure that the overall average monthly reward of the portfolio is
# at least 0.5% over the five-year investment period
    solver2.Add(sum(sum(portfolio_vars[timestamp, stock] * return_data[timestamp, stock] for
stock in stocks) for timestamp in timestamps[1:])\
                    / (len(timestamps) - 1) >= 1.005)
```

  - I created additional variables to determine the boundaries of risks to bounce_vars.

```
# Create these additional variables
    bounce_vars = {}
    for timestamp in timestamps[1:]:
        bounce_vars[timestamp] = solver2.NumVar(0, solver2.infinity(), "")
```

- And defined the bounding deviation constraints.

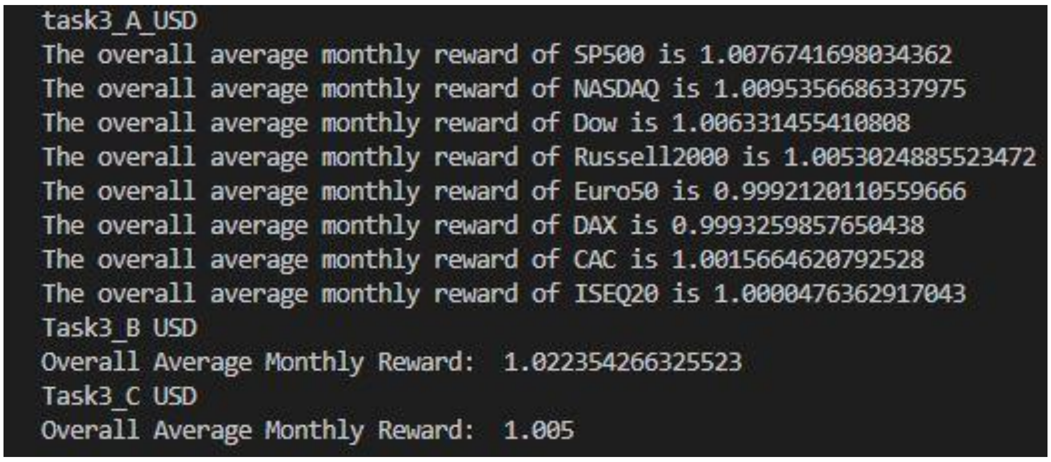
```
# implement the necessary constraints for bounding the deviation
    for timestamp in timestamps[1:]:
        solver2.Add(sum(portfolio_vars[timestamp, stock] * (return_data[timestamp, 
stock] - average_reward_data[stock]) for stock in stocks) \
                    >= -bounce_vars[timestamp])
        solver2.Add(sum(portfolio_vars[timestamp, stock] * (return_data[timestamp, 
stock] - average_reward_data[stock]) for stock in stocks) \
                    <= bounce_vars[timestamp])
```

- Finally, minimize the boundary vars.

```
solver2.Minimize(sum(bounce_vars[timestamp] for timestamp in timestamps[1:]))
```

B. Result

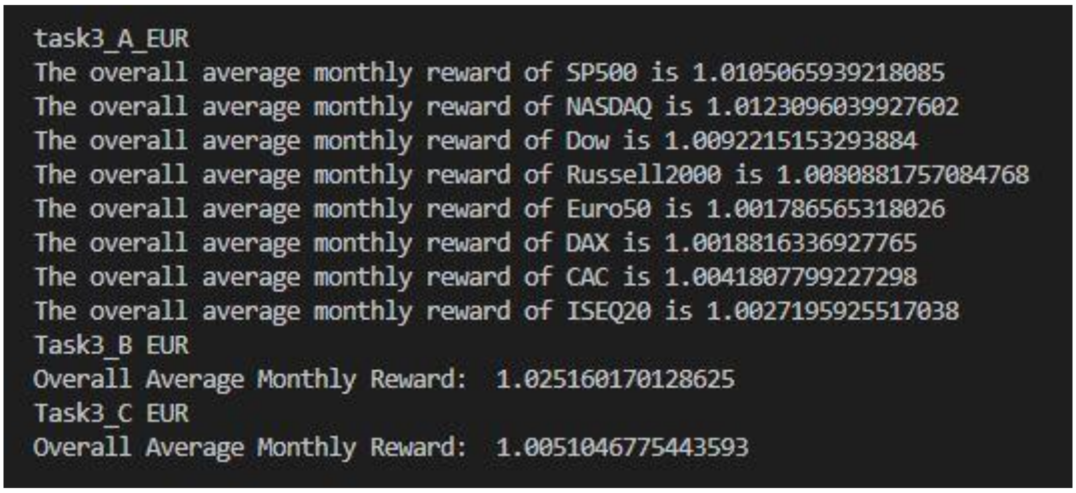
- USD result.



```
task3_A_USD
The overall average monthly reward of SP500 is 1.0076741698034362
The overall average monthly reward of NASDAQ is 1.0095356686337975
The overall average monthly reward of Dow is 1.006331455410808
The overall average monthly reward of Russell2000 is 1.0053024885523472
The overall average monthly reward of Euro50 is 0.9992120110559666
The overall average monthly reward of DAX is 0.9993259857650438
The overall average monthly reward of CAC is 1.0015664620792528
The overall average monthly reward of ISEQ20 is 1.0000476362917043
Task3_B USD
Overall Average Monthly Reward:  1.022354266325523
Task3_C USD
Overall Average Monthly Reward:  1.005
```

The position values are stored in task3_B_USD.csv, task3_C_USD.csv.

- EUR result.



```
task3_A_EUR
The overall average monthly reward of SP500 is 1.0105065939218085
The overall average monthly reward of NASDAQ is 1.0123096039927602
The overall average monthly reward of Dow is 1.0092215153293884
The overall average monthly reward of Russell2000 is 1.0080881757084768
The overall average monthly reward of Euro50 is 1.001786565318026
The overall average monthly reward of DAX is 1.0018816336927765
The overall average monthly reward of CAC is 1.0041807799227298
The overall average monthly reward of ISEQ20 is 1.0027195925517038
Task3_B EUR
Overall Average Monthly Reward:  1.025160170128625
Task3_C EUR
Overall Average Monthly Reward:  1.0051046775443593
```

The position values are stored in task3_B_EUR.csv, task3_C_EUR.csv.