R00182510 DA Assignment 2

Both Task 1 and Task 2 were implemented in a single file R00182510_DA_A2_Code.py

To execute Task_1, in the main() function please assign the value 'Task_1' to the execute variable.

To execute Task_2, in the main() function please assign the value 'Task_2' to the execute variable.

Task 1

In this task a Liner Programming model was developed to decide what raw material to order from which supplier, where to manufacture the products, and how to deliver the manufactured products to the customers so that the overall cost is minimized.

Subtask A:

Load Input data

In function Task1() of the code, Lines 7 to 54 shows the extraction of input data from excel file.

Subtask B:

Decision Variables

(i) Decision variables for the orders from the suppliers

```
supplier_orders[(supplier, material, factory)] = solver.NumVar(0, supp_stock.at[
                supplier, material], supplier + "_" + material + "_" + factory)
```

(ii) Decision variables for the production volume

```
prod_volume[(product, factory)] = solver.NumVar(0, prod_capacity.at[product,
                                  factory], product + "_" + factory)
```

(iii) Decision variables for the delivery to the customers

```
cust_delivery[(factory, product, customer)] = solver.NumVar(0,
     prod_capacity.at[product, factory], factory + "_" + product + "_" + customer)
```

In function Task1() of the code, Lines 58 to 83 implements these variables

Subtask C:

Constraint that ensure factories produce more than they ship to the customers.

```
for product in products:
    c = solver.Constraint(0, solver.infinity())
```

```
    for factory in factories:
        if prod_capacity.at[product, factory] != 0:
            c.SetCoefficient(prod_volume[(product, factory)], 1)
            for customer in customers:
                if cust_demand.at[product, customer] != 0 and
                 prod_capacity.at[product, factory] != 0:
                    c.SetCoefficient(cust_delivery[(factory, product, customer)],
                                                                                -1)
```

In function Task1() of the code, Lines 87 to 94 implements this constraint

Subtask D:

Constraint that ensure customer demand is met.

```
for customer in customers:
    for product in products:
        if cust_demand.at[product, customer] != 0:
            c = solver.Constraint(cust_demand.at[product, customer],
                                                cust_demand.at[product, customer])
            for factory in factories:
                if cust_demand.at[product, customer] != 0 and
                                    prod_capacity.at[product, factory] != 0:
                    c.SetCoefficient(cust_delivery[(factory, product, customer)],
                                                                                1)
```

In function Task1() of the code, Lines 97 to 103 implements this constraint

Subtask E:

Constraint that ensure suppliers have all ordered items in stock.

```
for material in materials:
    for supplier in suppliers:
        if supp_stock.at[supplier, material] != 0:
            c = solver.Constraint(0, supp_stock.at[supplier, material])
            for factory in factories:
                if (supplier, material, factory) in supplier_orders.keys():
                    c.SetCoefficient(supplier_orders[(supplier, material,
                                                                factory)], 1)
```

In function Task1() of the code, Lines 106 to 112 implements this constraint

Subtask F:

Constraint that ensure factories order enough materials to be able to manufacture all items

```
for material in materials:
    for factory in factories:
        c = solver.Constraint(0, 0)
```

```
        for product in products:
            if prod_req.at[product, material] != 0 and (product, factory) in
                                                        prod_volume.keys():
                c.SetCoefficient(prod_volume[(product, factory)], -
                                                    prod_req.at[product, material])
                for supplier in suppliers:
                    if (supplier, material, factory) in supplier_orders.keys():
                        c.SetCoefficient(supplier_orders[(supplier, material,
                                                                factory)], 1)
```

In function Task1() of the code, Lines 115 to 123 implements this constraint

Subtask G:

Constraint that ensure manufacturing capacities are not exceeded

```
for product in products:
    for factory in factories:
        if prod_capacity.at[product, factory] != 0:
            c = solver.Constraint(0, prod_capacity.at[product, factory])
            c.SetCoefficient(prod_volume[(product, factory)], 1)
```

In function Task1() of the code, Lines 126 to 130 implements this constraint

Subtask H:

Implement the Cost function including the supplier bills comprising shipping and material costs , the production cost of each factory and the cost of delivery to each customer.

```
cost = solver.Objective()
```

In function Task1() of the code, Lines 133 to 150 implements this function.

Subtask I:

Solve the linear program to determine the optimal overall cost

```
cost.SetMinimization()
status = solver.Solve()
```

Output:

```
Optimal solution found
```

In function Task1() of the code, Lines 153 to 154 implements this constraint

Subtask J:

For each factory how much material has to be ordered from each individual supplier

Output:

```
Supplier orders
Order from  Supplier D  to  Factory A  is  14.000000000000005 Material C
Order from  Supplier D  to  Factory A  is  50.0 Material D
Order from  Supplier E  to  Factory C  is  25.000000000000007 Material A
Order from  Supplier E  to  Factory B  is  4.000000000000003 Material A
Order from  Supplier E  to  Factory C  is  40.0 Material D
Order from  Supplier A  to  Factory A  is  20.0 Material B
Order from  Supplier A  to  Factory A  is  20.0 Material A
Order from  Supplier B  to  Factory C  is  6.000000000000002 Material B
Order from  Supplier B  to  Factory B  is  34.0 Material B
Order from  Supplier B  to  Factory A  is  4.000000000000002 Material B
Order from  Supplier B  to  Factory B  is  5.999999999999999 Material A
Order from  Supplier B  to  Factory A  is  19.000000000000004 Material A
Order from  Supplier C  to  Factory C  is  10.0 Material B
Order from  Supplier C  to  Factory C  is  20.0 Material C
Order from  Supplier C  to  Factory B  is  38.0 Material C
Order from  Supplier C  to  Factory C  is  34.99999999999999 Material D
Order from  Supplier C  to  Factory B  is  4.999999999999993 Material D
```

In function Task1() of the code, Lines 160 to 167 implements this question

Subtask K:

For each factory what is the supplier bill comprising material cost and delivery will be for each supplier

Output:

```
Supplier B  bill for  Factory B :  3500.0

Supplier B  bill for  Factory A :  2155.000000000001

Supplier B  bill for  Factory C :  1590.000000000001

Supplier C  bill for  Factory B :  10050.0

Supplier C  bill for  Factory C :  12450.0

Supplier A  bill for  Factory A :  2800.0

Supplier E  bill for  Factory B :  260.00000000000006

Supplier E  bill for  Factory C :  7325.0

Supplier D  bill for  Factory A :  6440.000000000001
```

In function Task1() of the code, Lines 169 to 190 implements this question

Subtask L:

For each factory how many units of each product are being manufactured and determine the total manufacturing cost for each individual factory

Output:

```
Products produced by each factory
1   Product B are produced by Factory B
4   Product C are produced by Factory B
2   Product A are produced by Factory B
3   Product D are produced by Factory A
1   Product B are produced by Factory A
6   Product A are produced by Factory A
5   Product D are produced by Factory C
2   Product A are produced by Factory C

Total manufacturing cost for  Factory B :  429.9999999999999

Total manufacturing cost for  Factory A :  1010.0000000000001

Total manufacturing cost for  Factory C :  425.0000000000001

Products delivered to each customer
2.0 Product A  items delivered to  Customer B  from  Factory B
1.0 Product A  items delivered to  Customer B  from  Factory A
3.0 Product D  items delivered to  Customer D  from  Factory A
1.0 Product D  items delivered to  Customer D  from  Factory C
4.0 Product C  items delivered to  Customer D  from  Factory B
1.0 Product D  items delivered to  Customer A  from  Factory C
```

In function Task1() of the code, Lines 192 to 207 implements this question

Subtask M:

For each customer how many units of each product are being shipped from each factory and determine the total shipping cost per customer.

Output:

```
Products delivered to each customer
2.0 Product A  items delivered to  Customer B  from  Factory B
1.0 Product A  items delivered to  Customer B  from  Factory A
3.0 Product D  items delivered to  Customer D  from  Factory A
1.0 Product D  items delivered to  Customer D  from  Factory C
4.0 Product C  items delivered to  Customer D  from  Factory B
1.0 Product D  items delivered to  Customer A  from  Factory C
7.0 Product A  items delivered to  Customer A  from  Factory C
3.0 Product D  items delivered to  Customer C  from  Factory A
2.0 Product B  items delivered to  Customer C  from  Factory A

Total shipping cost for  Customer B :  110.0

Total shipping cost for  Customer D :  240.0

Total shipping cost for  Customer A :  80.0

Total shipping cost for  Customer C :  150.0
```

In function Task1() of the code, Lines 210 to 226 implements this question

Subtask N:

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

Output:

```
3.0 Material A  required for making Product D  to  Customer A from Factory C
2.0 Material B  required for making Product D  to  Customer A from Factory C
15.0 Material D  required for making Product D  to  Customer A from Factory C
4.0 Material C  required for making Product D  to  Customer A from Factory C
35.0 Material A  required for making Product A  to  Customer A from Factory C
21.0 Material B  required for making Product A  to  Customer A from Factory C
3.0 Material A  required for making Product D  to  Customer D from Factory C
2.0 Material B  required for making Product D  to  Customer D from Factory C
15.0 Material D  required for making Product D  to  Customer D from Factory C
4.0 Material C  required for making Product D  to  Customer D from Factory C
28.0 Material B  required for making Product C  to  Customer D from Factory B
36.0 Material C  required for making Product C  to  Customer D from Factory B
10.0 Material A  required for making Product A  to  Customer B from Factory B
6.0 Material B  required for making Product A  to  Customer B from Factory B
9.0 Material A  required for making Product D  to  Customer D from Factory A
6.0 Material B  required for making Product D  to  Customer D from Factory A
45.0 Material D  required for making Product D  to  Customer D from Factory A
12.0 Material C  required for making Product D  to  Customer D from Factory A
5.0 Material A  required for making Product A  to  Customer B from Factory A
3.0 Material B  required for making Product A  to  Customer B from Factory A
10.0 Material D  required for making Product B  to  Customer C from Factory A
4.0 Material C  required for making Product B  to  Customer C from Factory A
9.0 Material A  required for making Product D  to  Customer C from Factory A
6.0 Material B  required for making Product D  to  Customer C from Factory A
45.0 Material D  required for making Product D  to  Customer C from Factory A
12.0 Material C  required for making Product D  to  Customer C from Factory A
```

In function Task1() of the code, Lines 228 to 238 implements this question

Task 2

The goal of this task is to develop and optimise an Integer Liner Programming model for allocating an arrival runway, a departure runway and a terminal for each flight so that the overall taxi distance of all planes is minimised.

Subtask A:

Load Input data

In function Task2() of the code, Lines 229 to 249 shows the extraction of input data from excel file.

Subtask B:

Decision variables

(i) Decision variables for the arrival runway allocation
(ii) Decision variables for the departure runway allocation

```
arrival_runway = {}
departure_runway = {}
for flight in flights:
    for runway in runways:
        arrival_runway[(flight, runway)] = solver.BoolVar("arrival_%s_%s" %
                                                (flight, runway))
        departure_runway[(flight, runway)] = solver.BoolVar("departure_%s_%s"
                                                % (flight, runway))
```

(iii) Decision variables for the terminal allocation

```
terminal_alloc = {}
for flight in flights:
    for terminal in terminals:
        terminal_alloc[(flight, terminal)] = solver.BoolVar("terminal_%s_%s" %
                                                (flight, terminal))
```

In function Task2() of the code, Lines 266 to 277 implements these variables

Subtask C:

Auxiliary variables for the taxi movements between runways and terminals for each flight.

```
arrival_runway_to_terminal = {}
for terminal in terminals:
    for runway in runways:
        for flight in flights:
            arrival_runway_to_terminal[(flight, terminal, runway)] =
                solver.BoolVar("arrival_%s_%s_%s" % (flight, terminal, runway))
```

```
departure_runway_to_terminal = {}
for terminal in terminals:
    for runway in runways:
        for flight in flights:
            departure_runway_to_terminal[(flight, terminal, runway)] =
                 solver.BoolVar("departure_%s_%s_%s" % (flight, terminal, runway))

for terminal in terminals:
    for runway in runways:
        total_flights_using_terminal_runway = solver.IntVar(0, solver.infinity(),
                                              "flight_count" + terminal + "_" + runway)
```

In function Task2(), lines 280 to 298 implements this constraint

Subtask D:

Every flight has exactly two taxi movements

```
for terminal in terminals:
    for runway in runways:
        c = solver.Constraint(1, 1)
        for flight in flights:
            c.SetCoefficient(departure_runway_to_terminal[(flight, terminal,
                                                           runway)], 1)

for terminal in terminals:
    for runway in runways:
        c = solver.Constraint(1, 1)
        for flight in flights:
            c.SetCoefficient(arrival_runway_to_terminal[(flight, terminal,
                                                         runway)], 1)
```

In function Task2(), lines 299 to 311 implements the above constraint

Subtask E and Subtask F:

Taxi movements of a flight are to and from the allocated terminal and the taxi movements of a flight include the allocated arrival and departure runways

```
for terminal in terminals:
    for runway in runways:
        for flight in flights:
            solver.Add(arrival_runway_to_terminal[(flight, terminal, runway)] >=
                       (arrival_runway[(flight, runway)] + terminal_alloc[(flight,
                                                           terminal)]) - 1)
            solver.Add(arrival_runway_to_terminal[(flight, terminal, runway)] <=
                                        arrival_runway[(flight, runway)])
            solver.Add(arrival_runway_to_terminal[(flight, terminal, runway)] <=
                                        terminal_alloc[(flight, terminal)])

            solver.Add(departure_runway_to_terminal[(flight, terminal, runway)] >=
                                        (departure_runway[(flight, runway)] +
```

```
                                        terminal_alloc[(flight, terminal)]) - 1)
            solver.Add(
                departure_runway_to_terminal[(flight, terminal, runway)] <=
                                        departure_runway[(flight, runway)])
            solver.Add(
                 departure_runway_to_terminal[(flight, terminal, runway)] <=
                                        terminal_alloc[(flight, terminal)])
```

In function Task2(), lines 312 to 329 implements this constraint

Subtask G:

Each flight has exactly one allocated arrival runway and exactly one allocated departure runway

```
# Subtask G. Each flight has exactly one allocated arrival runway and one
allocated departure runway
for flight in flights:
    c = solver.Constraint(1, 1)
    for runway in runways:
        c.SetCoefficient(arrival_runway[(flight, runway)], 1)

for flight in flights:
    c = solver.Constraint(1, 1)
    for runway in runways:
        c.SetCoefficient(departure_runway[(flight, runway)], 1)
```

In function Task2(), lines 330 to 340 implements this constraint

Subtask H:

Each flight is allocated to exactly one terminal

```
# Subtask H. Each flight is allocated to exactly one terminal
for flight in flights:
    c = solver.Constraint(1, 1)
    for terminal in terminal:
        c.SetCoefficient(departure_runway[(flight, runway)], 1)
```

In function Task2(), lines 341 to 346 implements this constraint

Subtask I:

No runway is used by more than one flight during each timeslot

```
# Subtask I. No runway is used by more than one flight during each timeslot
for runway in runways:
    for flight in flights:
        variables = []
        other_flights = [value for value in flights if value != flight]
        for other in other_flights:
            if flight_sched.at[flight, 'Arrival'] == flight_sched.at[other,
```

```
                                                                'Arrival']:
                variables.append(arrival_runway[(flight, runway)])
                variables.append(arrival_runway[(other, runway)])
            if flight_sched.at[flight, 'Arrival'] == flight_sched.at[other,
                                                                'Departure']:
                variables.append(arrival_runway[(flight, runway)])
                variables.append(departure_runway[(other, runway)])
            if flight_sched.at[flight, 'Departure'] == flight_sched.at[other,
                                                                'Arrival']:
                variables.append(departure_runway[(flight, runway)])
                variables.append(arrival_runway[(other, runway)])
            if flight_sched.at[flight, 'Departure'] == flight_sched.at[other,
                                                                'Departure']:
                variables.append(departure_runway[(flight, runway)])
                variables.append(departure_runway[(other, runway)])
        solver.Add(sum(variables) <= 1)
```

In function Task2(), lines 347 to 366 implements this constraint

Subtask J:

Terminal capacities are not exceeded

```
# Subtask J. Terminal capacities are not exceeded
for terminal in terminals:
    solver.Add(
        sum([terminal_alloc[(flight, terminal)] for flight in flights]) <=
                terminal_capacity.at[terminal, 'Gates'])
```

In function Task2(), lines 367 to 379 implements this constraint

Subtask K:

Implement the objective function

```
for terminal in terminals:
    for runway in runways:
        c = solver.Constraint(0, 0)
        for flight in flights:
            c.SetCoefficient(total_flights_using_terminal_runway, 1)
            c.SetCoefficient(arrival_runway_to_terminal[(flight, terminal,
                                                         runway)], -1)
            c.SetCoefficient(departure_runway_to_terminal[(flight, terminal,
                                                           runway)], -1)

# Subtask K: implement the objective function
cost = solver.Objective()
for terminal in terminals:
    for runway in runways:
        cost.SetCoefficient(total_flights_using_terminal_runway,
float(taxi_dist.at[runway, terminal]))

status = solver.Solve()
```

In function Task2(), lines 380 to 388 implements this constraint