Course Title: MGSC404: Foundations of Decision Analytics

Instructor: Setareh Farajollahzadeh

University: McGill University

Case Study (Deadline Feb 27)

Delivarables:

- 1- Jupyter Notebook,
- 2- Pdf of your jupyter notebook
- 3- Executive report : max 3 pages + 2 pages appendix (mathmathical formualtion of phases 1 and 2)

Section:

Group Number:

Members Names and studnet numbers:

- 1- Yannie Gao 261114710
- 2- Mario Oliveira 261254300
- 3-Zaid Al-Zubiedi 261135281
- 4- Omar Abouelmagd 261090403
- 5- Helena Mundstock 261072990

Packages

In [1]:
```
from gurobipy import *
from datascience import *
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plots
import pandas as pd
plots.style.use('ggplot')
```

Phase 1

Data/paramters/inputs/coefficients

In [3]:
```
Demand = [100, 50, 30, 80, 40]
Weight = [10, 8, 15, 5, 12]
Volume= [5,4,6,3,7]
```

Setup the model and decision variables

In [5]:
```
model = Model('Case Phase One')
X = {}
Y = {}
X = model.addVars(5, lb=0) # number of item i shipped on flight A for i = 0,1,2,3,4
Y = model.addVars(5, lb=0) # number of item i shipped on flight B for i = 0,1,2,3,4
```

```
Restricted license - for non-production use only - expires 2026-11-23
```

Constraints

In [7]:
```
Const = {}
for i in np.arange(5):
    Const[i] = model.addConstr(X[i]+Y[i]==Demand[i])# Demand constraints
Const[5] = model.addConstr( quicksum(X[i]*Volume[i] for i in range(5)) <=1000)# Vol
Const[6] = model.addConstr( quicksum(Y[i]*Volume[i] for i in range(5)) <=800)# Volu
Const[7] = model.addConstr( quicksum(X[i]*Weight[i] for i in range(5)) <=2000)# Wei
Const[8] = model.addConstr( quicksum(Y[i]*Weight[i] for i in range(5)) <=1500)# Wei
Const[9] = model.addConstr( Y[1]+Y[2] ==0)# Freezer Constraints, since those items
```

Run the optimization program

In [9]:
```
model.setObjective(2.5*(10*X[0]+8*X[1]+15*X[2]+5*X[3]+12*X[4])+2*(10*Y[0]+8*Y[1]+15

model.Params.LogToConsole = 0
model.optimize()
model.setParam('OutputFlag', 0)
```

```
Set parameter LogToConsole to value 0
```

Results of the optimization program

In [11]:
```
if model.status == GRB.OPTIMAL:
    print("Optimal solution found!")
    for i in range(5):
        print(f"X[{i}] = {X[i].X}, Y[{i}] = {Y[i].X}")
    print("Optimal Objective Value =", model.objVal)
else:
    print("No optimal solution found. Model status:", model.status)
```

```
Optimal solution found!
X[0] = 0.0, Y[0] = 100.0
X[1] = 50.0, Y[1] = 0.0
X[2] = 30.0, Y[2] = 0.0
X[3] = 76.0, Y[3] = 4.0
X[4] = 0.0, Y[4] = 40.0
Optimal Objective Value = 6075.0
```

Sensitivity Analysis

In [21]:
```
# Printing shadow prices (dual values)
print("Shadow Prices (Dual Values):")
for c in model.getConstrs():
    print(f"Constraint {c.ConstrName}: Shadow Price = {c.Pi:.4f}")
```

```python
# Printing optimal values of decision variables
print("\nOptimal Values of Decision Variables:")
for v in model.getVars():
    print(f"Variable {v.varName}: Optimal Value = {v.x:.4f}")

# Sensitivity Analysis: Allowable Increase/Decrease in RHS (Right-Hand Side)
print("\nSensitivity Analysis (Allowable Changes in Constraints):")
for c in model.getConstrs():
    print(f"Constraint {c.ConstrName}:")
    print(f"  RHS = {c.RHS:.4f}")

    try:  # Some solvers may not provide SARHSLow and SARHSUp
        print(f"  Minimum RHS = {c.SARHSLow:.4f}")
        print(f"  Allowable Decrease in RHS = {c.RHS - c.SARHSLow:.4f}")
        print(f"  Maximum RHS = {c.SARHSUp:.4f}")
        print(f"  Allowable Increase in RHS = {c.SARHSUp - c.RHS:.4f}")
    except AttributeError:
        print("  No sensitivity range available for this constraint.")
```

```
Shadow Prices (Dual Values):
Constraint R0: Shadow Price = 25.0000
Constraint R1: Shadow Price = 20.0000
Constraint R2: Shadow Price = 37.5000
Constraint R3: Shadow Price = 12.5000
Constraint R4: Shadow Price = 30.0000
Constraint R5: Shadow Price = 0.0000
Constraint R6: Shadow Price = 0.0000
Constraint R7: Shadow Price = 0.0000
Constraint R8: Shadow Price = -0.5000
Constraint R9: Shadow Price = 0.0000

Optimal Values of Decision Variables:
Variable C0: Optimal Value = 0.0000
Variable C1: Optimal Value = 50.0000
Variable C2: Optimal Value = 30.0000
Variable C3: Optimal Value = 76.0000
Variable C4: Optimal Value = 0.0000
Variable C5: Optimal Value = 100.0000
Variable C6: Optimal Value = 0.0000
Variable C7: Optimal Value = 0.0000
Variable C8: Optimal Value = 4.0000
Variable C9: Optimal Value = 40.0000

Sensitivity Analysis (Allowable Changes in Constraints):
Constraint R0:
  RHS = 100.0000
  Minimum RHS = 92.0000
  Allowable Decrease in RHS = 8.0000
  Maximum RHS = 102.0000
  Allowable Increase in RHS = 2.0000
Constraint R1:
  RHS = 50.0000
  Minimum RHS = 0.0000
  Allowable Decrease in RHS = 50.0000
  Maximum RHS = 146.2500
  Allowable Increase in RHS = 96.2500
Constraint R2:
  RHS = 30.0000
  Minimum RHS = 0.0000
  Allowable Decrease in RHS = 30.0000
  Maximum RHS = 81.3333
  Allowable Increase in RHS = 51.3333
Constraint R3:
  RHS = 80.0000
  Minimum RHS = 4.0000
  Allowable Decrease in RHS = 76.0000
  Maximum RHS = 210.6667
  Allowable Increase in RHS = 130.6667
Constraint R4:
  RHS = 40.0000
  Minimum RHS = 8.3333
  Allowable Decrease in RHS = 31.6667
  Maximum RHS = 41.6667
  Allowable Increase in RHS = 1.6667
Constraint R5:
  RHS = 1000.0000
  Minimum RHS = 608.0000
  Allowable Decrease in RHS = 392.0000
  Maximum RHS = inf
  Allowable Increase in RHS = inf
Constraint R6:
  RHS = 800.0000
  Minimum RHS = 792.0000
```

```
  Allowable Decrease in RHS = 8.0000
  Maximum RHS = inf
  Allowable Increase in RHS = inf
Constraint R7:
  RHS = 2000.0000
  Minimum RHS = 1230.0000
  Allowable Decrease in RHS = 770.0000
  Maximum RHS = inf
  Allowable Increase in RHS = inf
Constraint R8:
  RHS = 1500.0000
  Minimum RHS = 1480.0000
  Allowable Decrease in RHS = 20.0000
  Maximum RHS = 1513.3333
  Allowable Increase in RHS = 13.3333
Constraint R9:
  RHS = 0.0000
  Minimum RHS = 0.0000
  Allowable Decrease in RHS = 0.0000
  Maximum RHS = 0.0000
  Allowable Increase in RHS = 0.0000
```

In [23]:

```python
# Ensure the model is solved before extracting sensitivity analysis
if model.status != GRB.OPTIMAL:
    print("Optimizing Model...")
    model.optimize()

# Check if the model found an optimal solution
if model.status == GRB.OPTIMAL:

    print("\nModel solved successfully. Extracting sensitivity analysis...\n")

    # Extract shadow prices and constraint sensitivity data
    sensitivity_data = []

    for name, constraint in Const.items():
        sensitivity_data.append({
            "Constraint": name,
            "Shadow Price": constraint.Pi,  # Dual Value
            "RHS": constraint.RHS,  # Right-hand side value
            "Minimum RHS": constraint.SARHSLow,  # Lower bound before solution char
            "Allowable Decrease": constraint.RHS - constraint.SARHSLow,  # Allowed
            "Maximum RHS": constraint.SARHSUp,  # Upper bound before solution chang
            "Allowable Increase": constraint.SARHSUp - constraint.RHS  # Allowed ir
        })

    # Convert sensitivity data to a pandas DataFrame
    sensitivity_df = pd.DataFrame(sensitivity_data)

    # Extract optimal values of decision variables
    variable_data = []

    for v in model.getVars():
        variable_data.append({"Variable": v.varName, "Optimal Value": v.X})

    # Convert variable data to a pandas DataFrame
    variable_df = pd.DataFrame(variable_data)

    # Display both tables in sequence
    print("\nConstraint Sensitivity Analysis Table:")
    display(sensitivity_df)

    print("\nOptimal Variable Values Table:")
    display(variable_df)
```

```
else:
    print("\nModel is not optimal. Please check for infeasibility or errors.")
```

```
Model solved successfully. Extracting sensitivity analysis...


Constraint Sensitivity Analysis Table:
```

	Constraint	Shadow Price	RHS	Minimum RHS	Allowable Decrease	Maximum RHS	Allowable Increase
0	0	25.0	100.0	92.000000	8.000000	102.000000	2.000000
1	1	20.0	50.0	0.000000	50.000000	146.250000	96.250000
2	2	37.5	30.0	0.000000	30.000000	81.333333	51.333333
3	3	12.5	80.0	4.000000	76.000000	210.666667	130.666667
4	4	30.0	40.0	8.333333	31.666667	41.666667	1.666667
5	5	0.0	1000.0	608.000000	392.000000	inf	inf
6	6	0.0	800.0	792.000000	8.000000	inf	inf
7	7	0.0	2000.0	1230.000000	770.000000	inf	inf
8	8	-0.5	1500.0	1480.000000	20.000000	1513.333333	13.333333
9	9	0.0	0.0	0.000000	0.000000	0.000000	0.000000

```
Optimal Variable Values Table:
```

	Variable	Optimal Value
0	C0	0.0
1	C1	50.0
2	C2	30.0
3	C3	76.0
4	C4	0.0
5	C5	100.0
6	C6	0.0
7	C7	0.0
8	C8	4.0
9	C9	40.0

Phase 2

Data/paramters/inputs/coefficients

In [27]:

```
Supply = ((70,70,60,50,300),(0,0,0,30,0),(0,0,60,0,0))
Demand = ((0,0,0,0,0),(30,20,120,0,100),(40,50,0,80,200))
Weight = [10, 8, 15, 5, 12]
Volume= [5,4,6,3,7]
```

Setup the model and decision variables

In [30]:
```
model = Model('Cargo Allocation: Phase Two')

A={}
B={}
C={}
D={}
E={}

A = model.addVars(5, lb = 0) # number of item i+1 shipped on route A for i = 0,1,2,
B = model.addVars(5, lb = 0) # number of item i+1 shipped on route B for i = 0,1,2,
C = model.addVars(5, lb = 0) # number of item i+1 shipped on route C for i = 0,1,2,
D = model.addVars(5, lb = 0) # number of item i+1 shipped on route D for i = 0,1,2,
E = model.addVars(5, lb = 0) # number of item i+1 shipped on route E for i = 0,1,2,
```

Constraints

In [33]:
```
#Adding Constraints
Const = {}

#Supply Constraints
for i in np.arange(5):
    Const[i] = model.addConstr(A[i]+B[i]+D[i] == Supply[0][i]) #Products leaving Mo
for i in np.arange(5):
    Const[5+i] = model.addConstr(C[i] <= Supply[1][i]+B[i])#Products leaving Toront
for i in np.arange(5):
    Const[10+i] = model.addConstr(E[i] <= Supply[2][i]+A[i]+C[i]+D[i])#Products lea

#Demmand Constraints
for i in np.arange(5):
    Const[15+i] = model.addConstr(Supply[1][i]+B[i]+E[i]-C[i] == Demand[1][i])# Tor
for i in np.arange(5):
    Const[20+i] = model.addConstr(Supply[2][i]+A[i]+C[i]+D[i]-E[i] == Demand[2][i])

#Weight Constraints
Const[25] = model.addConstr(quicksum(A[i]*Weight[i] for i in range(5)) <= 6000)
Const[26] = model.addConstr(quicksum(B[i]*Weight[i] for i in range(5)) <= 6800)
Const[27] = model.addConstr(quicksum(C[i]*Weight[i] for i in range(5)) <= 5700)
Const[28] = model.addConstr(quicksum(D[i]*Weight[i] for i in range(5)) <= 7000)
Const[29] = model.addConstr(quicksum(E[i]*Weight[i] for i in range(5)) <= 9400)

#Volume Constraints
Const[30] = model.addConstr(quicksum(A[i]*Volume[i] for i in range(5)) <= 5000)
Const[31] = model.addConstr(quicksum(B[i]*Volume[i] for i in range(5)) <= 7000)
Const[32] = model.addConstr(quicksum(C[i]*Volume[i] for i in range(5)) <= 7350)
Const[33] = model.addConstr(quicksum(D[i]*Volume[i] for i in range(5)) <= 7800)
Const[34] = model.addConstr(quicksum(E[i]*Volume[i] for i in range(5)) <= 8500)

#Freezer Constraints
Const[35] = model.addConstr(B[1] + B[2] + D[1] + D[2] == 0)
```

Run the optimization program

In [36]:
```
model.setObjective(A[0]*18 + A[1]*14.4 + A[2]*27 + A[3]*9 + A[4]*21.6
                   + B[0]*25 + B[1]*20 + B[2]*37.5 + B[3]*12.5 + B[4]*30
                   + C[0]*23 + C[1]*18.4 + C[2]*34.5 + C[3]*11.5 + C[4]*27.6
                   + D[0]*20 + D[1]*16 + D[2]*30 + D[3]*10 + D[4]*24
```

```
                         + E[0]*30 + E[1]*24 + E[2]*45 + E[3]*15 + E[4]*36, GRB.MINIMIZE)

#Run the Model
model.optimize()
```

```
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (win64 - Windows 11.0 (26100.2))

CPU model: 13th Gen Intel(R) Core(TM) i9-13900H, instruction set [SSE2|AVX|AVX2]
Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 36 rows, 25 columns and 134 nonzeros
Model fingerprint: 0xf0e5924b
Coefficient statistics:
  Matrix range     [1e+00, 2e+01]
  Objective range  [9e+00, 5e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+01, 9e+03]
Presolve removed 30 rows and 17 columns
Presolve time: 0.02s
Presolved: 6 rows, 8 columns, 15 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.7043000e+04   1.656250e+01   0.000000e+00      0s
       2    1.8093000e+04   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.03 seconds (0.00 work units)
Optimal objective  1.809300000e+04
```

Results of the optimization program

In [39]:
```
if model.status == GRB.OPTIMAL:
    print("Optimal solution found!")
    for i in range(5):
        print(f"A[{i}] = {A[i].X}, B[{i}] = {B[i].X}, C[{i}] = {C[i].X}, D[{i}] = {
    print("Optimal Objective Value =", model.objVal)
else:
    print("No optimal solution found. Model status:", model.status)
```

```
Optimal solution found!
A[0] = 40.0, B[0] = 30.0, C[0] = 0.0, D[0] = 0.0, E[0] = 0.0
A[1] = 70.0, B[1] = 0.0, C[1] = 0.0, D[1] = 0.0, E[1] = 20.0
A[2] = 60.0, B[2] = 0.0, C[2] = 0.0, D[2] = 0.0, E[2] = 120.0
A[3] = 50.0, B[3] = 0.0, C[3] = 30.0, D[3] = 0.0, E[3] = 0.0
A[4] = 200.0, B[4] = 100.0, C[4] = 0.0, D[4] = 0.0, E[4] = 0.0
Optimal Objective Value = 18093.0
```

Sensitivity Analysis Table

In [45]:
```
# Ensure the model is solved before extracting sensitivity analysis
if model.status != GRB.OPTIMAL:
    print("Optimizing Model...")
    model.optimize()

# Check if the model found an optimal solution
if model.status == GRB.OPTIMAL:

    print("\nModel solved successfully. Extracting sensitivity analysis...\n")

    # Extract shadow prices and constraint sensitivity data
    sensitivity_data = []
```

```python
    for name, constraint in Const.items():
        sensitivity_data.append({
            "Constraint": name,
            "Shadow Price": constraint.Pi,  # Dual Value
            "RHS": constraint.RHS,  # Right-hand side value
            "Minimum RHS": constraint.SARHSLow,  # Lower bound before solution char
            "Allowable Decrease": constraint.RHS - constraint.SARHSLow,  # Allowed
            "Maximum RHS": constraint.SARHSUp,  # Upper bound before solution chang
            "Allowable Increase": constraint.SARHSUp - constraint.RHS  # Allowed ir
        })

    # Convert sensitivity data to a pandas DataFrame
    sensitivity_df = pd.DataFrame(sensitivity_data)

    # Extract optimal values of decision variables
    variable_data = []

    for v in model.getVars():
        variable_data.append({"Variable": v.varName, "Optimal Value": v.X})

    # Convert variable data to a pandas DataFrame
    variable_df = pd.DataFrame(variable_data)

    # Display both tables in sequence
    print("\nConstraint Sensitivity Analysis Table:")
    display(sensitivity_df)

    print("\nOptimal Variable Values Table:")
    display(variable_df)

else:
    print("\nModel is not optimal. Please check for infeasibility or errors.")
```

```
Model solved successfully. Extracting sensitivity analysis...


Constraint Sensitivity Analysis Table:
```

	Constraint	Shadow Price	RHS	Minimum RHS	Allowable Decrease	Maximum RHS	Allowable Increase
0	0	18.0	70.0	70.0	0.0	70.0	0.0
1	1	38.4	70.0	70.0	0.0	70.0	0.0
2	2	72.0	60.0	60.0	0.0	60.0	0.0
3	3	9.0	50.0	50.0	0.0	50.0	0.0
4	4	21.6	300.0	300.0	0.0	300.0	0.0
5	5	0.0	0.0	-30.0	30.0	inf	inf
6	6	0.0	0.0	0.0	0.0	inf	inf
7	7	0.0	0.0	0.0	0.0	inf	inf
8	8	0.0	30.0	30.0	0.0	inf	inf
9	9	0.0	0.0	-100.0	100.0	inf	inf
10	10	0.0	0.0	-40.0	40.0	inf	inf
11	11	0.0	0.0	-50.0	50.0	inf	inf
12	12	0.0	60.0	60.0	0.0	inf	inf
13	13	0.0	0.0	-80.0	80.0	inf	inf
14	14	0.0	0.0	-200.0	200.0	inf	inf
15	15	7.0	30.0	30.0	0.0	30.0	0.0
16	16	0.0	20.0	20.0	0.0	20.0	0.0
17	17	0.0	120.0	120.0	0.0	120.0	0.0
18	18	-11.5	-30.0	-30.0	0.0	-30.0	0.0
19	19	8.4	100.0	100.0	0.0	100.0	0.0
20	20	0.0	40.0	40.0	0.0	40.0	0.0
21	21	-24.0	50.0	50.0	0.0	50.0	0.0
22	22	-45.0	-60.0	-60.0	0.0	-60.0	0.0
23	23	0.0	80.0	80.0	0.0	80.0	0.0
24	24	0.0	200.0	200.0	0.0	200.0	0.0
25	25	0.0	6000.0	4510.0	1490.0	inf	inf
26	26	0.0	6800.0	1500.0	5300.0	inf	inf
27	27	0.0	5700.0	150.0	5550.0	inf	inf
28	28	0.0	7000.0	0.0	7000.0	inf	inf
29	29	0.0	9400.0	1960.0	7440.0	inf	inf
30	30	0.0	5000.0	2390.0	2610.0	inf	inf
31	31	0.0	7000.0	850.0	6150.0	inf	inf
32	32	0.0	7350.0	90.0	7260.0	inf	inf
33	33	0.0	7800.0	0.0	7800.0	inf	inf
34	34	0.0	8500.0	800.0	7700.0	inf	inf

	Constraint	Shadow Price	RHS	Minimum RHS	Allowable Decrease	Maximum RHS	Allowable Increase
35	35	-34.5	0.0	0.0	0.0	60.0	60.0

```
Optimal Variable Values Table:
```

	Variable	Optimal Value
0	C0	40.0
1	C1	70.0
2	C2	60.0
3	C3	50.0
4	C4	200.0
5	C5	30.0
6	C6	0.0
7	C7	0.0
8	C8	0.0
9	C9	100.0
10	C10	0.0
11	C11	0.0
12	C12	0.0
13	C13	30.0
14	C14	0.0
15	C15	0.0
16	C16	0.0
17	C17	0.0
18	C18	0.0
19	C19	0.0
20	C20	0.0
21	C21	20.0
22	C22	120.0
23	C23	0.0
24	C24	0.0

Phase Two section C

New Model and Decision Variables

In [49]:

```
model = Model('Cargo Allocation: Phase Two - Part C')

A = model.addVars(5, lb=0)  # Route A (M → V)
B = model.addVars(5, lb=0)  # Route B (M → T)
```

```
C = model.addVars(5, lb=0)  # Route C (T → V)
D = model.addVars(5, lb=0)  # Route D (M → V)
E = model.addVars(5, lb=0)  # Route E (V → T)
M = model.addVar(name = 'New Montreal Supply for Item 3')
V = model.addVar(name = 'New Vancouver Supply for Item 3')
```

New Coeficients

In [52]:
```
Supply = ((70, 70, 60+M, 50, 300), (0, 0, 0, 30, 0), (0, 0, 60+V, 0, 0))
Demand = ((0, 0, 0, 0, 0), (30, 20, 220, 0, 100), (40, 50, 0, 80, 200))  # Updated
Weight = [10, 8, 15, 5, 12]
Volume = [5, 4, 6, 3, 7]
```

New Constraints

In [55]:
```
# Adding Constraints
Const = {}

# Supply Constraints
for i in range(5):
    Const[i] = model.addConstr(A[i] + B[i] + D[i] == Supply[0][i])  # Products Leav
    Const[5 + i] = model.addConstr(C[i] <= Supply[1][i] + B[i])  # Products Leaving
    Const[10 + i] = model.addConstr(E[i] <= Supply[2][i] + A[i] + C[i] + D[i])  # P

# **Updated Demand Constraints**
for i in range(5):
    Const[15 + i] = model.addConstr(Supply[1][i] + B[i] + E[i] - C[i] == Demand[1][
    Const[20 + i] = model.addConstr(Supply[2][i] + A[i] + C[i] + D[i] - E[i] == Den

# Weight Constraints
Const[25] = model.addConstr(sum(A[i] * Weight[i] for i in range(5)) <= 6000)
Const[26] = model.addConstr(sum(B[i] * Weight[i] for i in range(5)) <= 6800)
Const[27] = model.addConstr(sum(C[i] * Weight[i] for i in range(5)) <= 5700)
Const[28] = model.addConstr(sum(D[i] * Weight[i] for i in range(5)) <= 7000)
Const[29] = model.addConstr(sum(E[i] * Weight[i] for i in range(5)) <= 9400)
# Volume Constraints
Const[30] = model.addConstr(sum(A[i] * Volume[i] for i in range(5)) <= 5000)
Const[31] = model.addConstr(sum(B[i] * Volume[i] for i in range(5)) <= 7000)
Const[32] = model.addConstr(sum(C[i] * Volume[i] for i in range(5)) <= 7350)
Const[33] = model.addConstr(sum(D[i] * Volume[i] for i in range(5)) <= 7800)
Const[34] = model.addConstr(sum(E[i] * Volume[i] for i in range(5)) <= 8500)

# Freezer Constraints
Const[35] = model.addConstr(B[1] + B[2] + D[1] + D[2] == 0)
```

Optimization

In [58]:
```
model.setObjective(
    A[0] * 18 + A[1] * 14.4 + A[2] * 27 + A[3] * 9 + A[4] * 21.6 +
    B[0] * 25 + B[1] * 20 + B[2] * 37.5 + B[3] * 12.5 + B[4] * 30 +
    C[0] * 23 + C[1] * 18.4 + C[2] * 34.5 + C[3] * 11.5 + C[4] * 27.6 +
    D[0] * 20 + D[1] * 16 + D[2] * 30 + D[3] * 10 + D[4] * 24 +
    E[0] * 30 + E[1] * 24 + E[2] * 45 + E[3] * 15 + E[4] * 36,
    GRB.MINIMIZE
)

# Run the Model
model.optimize()
```

```
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (win64 - Windows 11.0 (26100.2))

CPU model: 13th Gen Intel(R) Core(TM) i9-13900H, instruction set [SSE2|AVX|AVX2]
Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 36 rows, 27 columns and 137 nonzeros
Model fingerprint: 0xc95bc0da
Coefficient statistics:
  Matrix range     [1e+00, 2e+01]
  Objective range  [9e+00, 5e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+01, 9e+03]
Presolve removed 30 rows and 19 columns
Presolve time: 0.06s
Presolved: 6 rows, 8 columns, 15 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.1543000e+04   1.656250e+01   0.000000e+00      0s
       2    2.2593000e+04   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.11 seconds (0.00 work units)
Optimal objective  2.259300000e+04
```

Results

In [61]:

```python
# Print the results
if model.status == GRB.OPTIMAL:
    print("\nOptimal solution found!")

    for i in range(5):
        print(f"A[{i}] = {A[i].X}, B[{i}] = {B[i].X}, C[{i}] = {C[i].X}, D[{i}] = {
    print(f"M = {M.X}, V = {V.X}")

    print("Optimal Objective Value =", model.objVal)

else:
    print("\nModel did not find an optimal solution. Please check for infeasibility
```

```
Optimal solution found!
A[0] = 40.0, B[0] = 30.0, C[0] = 0.0, D[0] = 0.0, E[0] = 0.0
A[1] = 70.0, B[1] = 0.0, C[1] = 0.0, D[1] = 0.0, E[1] = 20.0
A[2] = 60.0, B[2] = 0.0, C[2] = 0.0, D[2] = 0.0, E[2] = 220.0
A[3] = 50.0, B[3] = 0.0, C[3] = 30.0, D[3] = 0.0, E[3] = 0.0
A[4] = 200.0, B[4] = 100.0, C[4] = 0.0, D[4] = 0.0, E[4] = 0.0
M = 0.0, V = 100.0
Optimal Objective Value = 22593.0
```