

0. Imports and Setting up Anthropic API Client

```
from google.colab import drive
```

```
drive.mount('/content/drive')
```

Mounted at /content/drive

```
!pip install python-dotenv
```

```
import os
import dotenv
```

```
dotenv.load_dotenv('/content/drive/MyDrive/.env')
```

Collecting python-dotenv
 Downloading python_dotenv-1.0.1-py3-none-any.whl (19 kB)
 Installing collected packages: python-dotenv
 Successfully installed python-dotenv-1.0.1
 True

```
# Load Prompts and Problem Description
```

```
prompt1_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt1_MathematicalModel.txt'
```

```
prompt2_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt2_PyomoCode.txt'
```

```
problem_desc_path = '/content/drive/MyDrive/Thesis/ProblemDescriptions/LP/LP3.txt'
```

```
prompt1_file = open(prompt1_path, "r")
```

```
prompt2_file = open(prompt2_path, "r")
```

```
problem_desc_file = open(problem_desc_path, "r")
```

```
prompt1 = prompt1_file.read()
```

```
print("Prompt 1:\n", prompt1)
```

```
prompt2 = prompt2_file.read()
```

```
print("Prompt 2:\n", prompt2)
```

```
problem_desc = problem_desc_file.read()
```

```
print("Problem Description:\n", problem_desc)
```

Prompt 1:
 Please write a mathematical optimization model for this problem. Include parameters, decision variables, the objective
 Prompt 2:
 Please write a python pyomo code for this optimization problem.
 Use sample data where needed.
 Indicate where you use sample data.
 Problem Description:
 The PRODA, S.A. industrial products firm has to face the problem of scheduling the weekly production of its three products (P1, P2 and P3). These products are sold to large industrial firms and PRODA, S.A. wishes to supply its products in quantities that are more profitable for it.

Each product entails three operations contributing to the costs: smelting; mechanisation; assembly and packaging. The smelting operations for products P1 and P2 could be subcontracted, but the smelting operation for product preventing the use of subcontracts. PRODA also want to know, how much they should subcontract.

For product P1 the direct unit costs of all possible operations are:

- smelting at PRODA: 0.30\$
- subcontracted smelting: 0.50\$
- mechanisation: 0.20\$
- Assembly and packaging: 0.3\$

The unit sales price is 1.50\$.

For product P2 the direct unit costs of all possible operations are:

- smelting at PRODA: 0.50\$
- subcontracted smelting: 0.60\$
- mechanisation: 0.10\$
- Assembly and packaging: 0.20\$

The unit sales price is 1.80\$.

For product P3 the direct unit costs of all possible operations are:

- smelting at PRODA: 0.40\$
- mechanisation: 0.27\$
- Assembly and packaging: 0.20\$

The unit sales price is 1.97\$.

Each unit of product P1 requires 6 min of smelting time (if performed at PRODA, S.A.), 6 min of mechanisation time and 3 The objective is to maximize weekly profits.

```
!pip install anthropic
```

 Show hidden output

```
# Importing Anthropic & Setting Headers
import anthropic


client = anthropic.Anthropic(
    # defaults to os.environ.get("ANTHROPIC_API_KEY")
    api_key=os.environ.get("ANTHROPIC_API_KEY"),
)
```

✓ 1. Prompt 1: Create Mathematical Model

```
message = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt1,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)
```

```
response_p1 = message.content[0].text
```

```
# Print response
print(response_p1)
```

 Parameters:

- i: product index (i = 1, 2, 3)
- pi: unit sales price of product i
- ci: unit cost of smelting product i at PRODA
- si: unit cost of subcontracted smelting for product i
- mi: unit cost of mechanisation for product i
- ai: unit cost of assembly and packaging for product i
- ti: smelting time required per unit of product i at PRODA (in minutes)
- ui: mechanisation time required per unit of product i (in minutes)
- vi: assembly and packaging time required per unit of product i (in minutes)
- T: total available smelting time at PRODA per week (in minutes)
- U: total available mechanisation time per week (in minutes)
- V: total available assembly and packaging time per week (in minutes)

Decision variables:

- xi: quantity of product i to produce
- yi: quantity of product i to subcontract for smelting

Objective function:

$$\text{Maximize } Z = (p_1 - c_1)x_1 + (p_2 - c_2)x_2 + (p_3 - c_3)x_3 - s_1y_1 - s_2y_2 - m_1x_1 - m_2x_2 - m_3x_3 - a_1x_1 - a_2x_2 - a_3x_3$$

Constraints:

1. Smelting time constraint at PRODA:

$$t_1(x_1 - y_1) + t_2(x_2 - y_2) + t_3x_3 \leq T$$
2. Mechanisation time constraint:

$$u_1x_1 + u_2x_2 + u_3x_3 \leq U$$
3. Assembly and packaging time constraint:

$$v_1x_1 + v_2x_2 + v_3x_3 \leq V$$
4. Non-negativity constraints:

$$x_i \geq 0, y_i \geq 0 \text{ for } i = 1, 2, 3$$
5. Subcontracting constraints:

$$y_1 \leq x_1$$

$$y_2 \leq x_2$$

$$y_3 = 0 \text{ (since smelting for product P3 cannot be subcontracted)}$$

The objective function maximizes the total profit by considering the revenue from selling the products and subtracting t

✓ 2. Prompt 2: Write Code based on Mathematical Model

```

message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": response_p1}
    ]
)

response_p2 = message2.content[0].text

# Print response
print(response_p2)

model.v = {1: 3, 2: 4, 3: 5}      # Assembly and packaging time per unit
model.T = 2400                    # Total available smelting time per week
model.U = 1200                    # Total available mechanisation time per week
model.V = 900                    # Total available assembly and packaging time per week

# Define the decision variables
model.x = Var(model.I, domain=NonNegativeReals)
model.y = Var(model.I, domain=NonNegativeReals)

# Define the objective function
def obj_rule(model):
    return sum((model.p[i] - model.c[i]) * model.x[i] for i in model.I) - \
        sum(model.s[i] * model.y[i] for i in model.I if i != 3) - \
        sum(model.m[i] * model.x[i] for i in model.I) - \
        sum(model.a[i] * model.x[i] for i in model.I)
model.obj = Objective(rule=obj_rule, sense=maximize)

# Define the constraints
def smelting_time_constraint(model):
    return sum(model.t[i] * (model.x[i] - model.y[i]) for i in model.I if i != 3) + model.t[3] * model.x[3] <= model.T
model.smelting_time = Constraint(rule=smelting_time_constraint)

def mechanisation_time_constraint(model):
    return sum(model.u[i] * model.x[i] for i in model.I) <= model.U
model.mechanisation_time = Constraint(rule=mechanisation_time_constraint)

def assembly_packaging_time_constraint(model):
    return sum(model.v[i] * model.x[i] for i in model.I) <= model.V
model.assembly_packaging_time = Constraint(rule=assembly_packaging_time_constraint)

def subcontracting_constraint_1(model):
    return model.y[1] <= model.x[1]
model.subcontracting_1 = Constraint(rule=subcontracting_constraint_1)

def subcontracting_constraint_2(model):
    return model.y[2] <= model.x[2]
model.subcontracting_2 = Constraint(rule=subcontracting_constraint_2)

def subcontracting_constraint_3(model):
    return model.y[3] == 0
model.subcontracting_3 = Constraint(rule=subcontracting_constraint_3)

# Solve the model
solver = SolverFactory('glpk')
results = solver.solve(model)

# Print the results
print("Objective value:", value(model.obj))
print("Optimal solution:")
for i in model.I:
    print("Product", i)
    print("  Quantity to produce (x):", value(model.x[i]))
    print("  Quantity to subcontract (y):", value(model.y[i]))

```

In this code, I used sample data for the parameters such as unit sales price, unit costs, and time requirements. You can modify these values to match your specific problem data.

The code creates a Pyomo model, defines the sets, parameters, decision variables, objective function, and constraints, and then solves the model using the GLPK solver.

4. Input Problem Data and Test Model Code

```

# Installing pyomo and solver
!pip install -q pyomo
!apt-get install -y -qq glpk-utils
!pip install glpk

```

Show hidden output

```
from pyomo.environ import *
```

```

# Create a Pyomo model
model = ConcreteModel()

# Define the sets
model.I = RangeSet(1, 3)

# Define the parameters (using sample data)
model.p = {1: 1.5, 2: 1.8, 3: 1.97} # Unit sales price
model.c = {1: 0.3, 2: 0.5, 3: 0.4}   # Unit cost of smelting at PRODA
model.s = {1: 0.5, 2: 0.6}           # Unit cost of subcontracted smelting
model.m = {1: 0.2, 2: 0.1, 3: 0.27} # Unit cost of mechanisation
model.a = {1: 0.3, 2: 0.2, 3: 0.2}   # Unit cost of assembly and packaging
model.t = {1: 6, 2: 10, 3: 8}        # Smelting time per unit at PRODA
model.u = {1: 6, 2: 3, 3: 8}         # Mechanisation time per unit
model.v = {1: 3, 2: 2, 3: 2}         # Assembly and packaging time per unit
model.T = 8000                        # Total available smelting time per week
model.U = 12000                       # Total available mechanisation time per week
model.V = 10000                       # Total available assembly and packaging time per week

# Define the decision variables
model.x = Var(model.I, domain=NonNegativeReals)
model.y = Var(model.I, domain=NonNegativeReals)

# Define the objective function
def obj_rule(model):
    return sum((model.p[i] - model.c[i]) * model.x[i] for i in model.I) - \
           sum(model.s[i] * model.y[i] for i in model.I if i != 3) - \
           sum(model.m[i] * model.x[i] for i in model.I) - \
           sum(model.a[i] * model.x[i] for i in model.I)
model.obj = Objective(rule=obj_rule, sense=maximize)

# Define the constraints
def smelting_time_constraint(model):
    return sum(model.t[i] * (model.x[i] - model.y[i]) for i in model.I if i != 3) + model.t[3] * model.x[3] <= model.T
model.smelting_time = Constraint(rule=smelting_time_constraint)

def mechanisation_time_constraint(model):
    return sum(model.u[i] * model.x[i] for i in model.I) <= model.U
model.mechanisation_time = Constraint(rule=mechanisation_time_constraint)

def assembly_packaging_time_constraint(model):
    return sum(model.v[i] * model.x[i] for i in model.I) <= model.V
model.assembly_packaging_time = Constraint(rule=assembly_packaging_time_constraint)

def subcontracting_constraint_1(model):
    return model.y[1] <= model.x[1]
model.subcontracting_1 = Constraint(rule=subcontracting_constraint_1)

def subcontracting_constraint_2(model):
    return model.y[2] <= model.x[2]
model.subcontracting_2 = Constraint(rule=subcontracting_constraint_2)

def subcontracting_constraint_3(model):
    return model.y[3] == 0
model.subcontracting_3 = Constraint(rule=subcontracting_constraint_3)

# Solve the model
solver = SolverFactory('glpk')
results = solver.solve(model)

# Print the results
print("Objective value:", value(model.obj))
print("Optimal solution:")
for i in model.I:
    print("Product", i)
    print("  Quantity to produce (x):", value(model.x[i]))
    print("  Quantity to subcontract (y):", value(model.y[i]))

↗ Objective value: 2080.0
Optimal solution:
Product 1
  Quantity to produce (x): 0.0
  Quantity to subcontract (y): 0.0
Product 2
  Quantity to produce (x): 4000.0
  Quantity to subcontract (y): 3200.0
Product 3
  Quantity to produce (x): 0.0
  Quantity to subcontract (y): 0.0

```

✓ 5. Correct The Model Code to Test Mathematical Model (if applicable)