

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
```

```
# Variables Prompt
```

```
prompt11_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt11_MathematicalModel.txt'
```

```
# Objective Prompt
```

```
prompt12_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt12_MathematicalModel.txt'
```

```
# Constraint Prompt
```

```
prompt13_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt13_MathematicalModel.txt'
```

```
# Code Prompt
```

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

```
problem_desc_path = '/content/drive/MyDrive/Thesis/ProblemDescriptions/LP/LP1.txt'
```

```
prompt11_file = open(prompt11_path, "r")
prompt12_file = open(prompt12_path, "r")
prompt13_file = open(prompt13_path, "r")
prompt2_file = open(prompt2_path, "r")
problem_desc_file = open(problem_desc_path, "r")
```

```
prompt11 = prompt11_file.read()
print("Prompt 1.1 (Variables):\n", prompt11)
```

```
prompt12 = prompt12_file.read()
print("Prompt 1.2 (Objective):\n", prompt12)
```

```
prompt13 = prompt13_file.read()
print("Prompt 1.3 (Constraints):\n", prompt13)
```

```
prompt2 = prompt2_file.read()
print("Prompt 2:\n", prompt2)
```

```
problem_desc = problem_desc_file.read()
print("Problem Description:\n", problem_desc)
```

Prompt 1.1 (Variables):
 Please formulate only the variables for this mathematical optimization problem.
 Prompt 1.2 (Objective):
 Please formulate only the objective function for this mathematical optimization problem.
 Prompt 1.3 (Constraints):
 Please formulate only the constraints for this mathematical optimization problem.
 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:
 A firm from Milan sells chemical products for professional cosmetics. It is planning the production of three products, time by mixing two different components: C1 and C2. All the end products must contain at least one of the two components, and not necessarily both.

For the next planning period, 10,000 l of C1 and 15,000 l of C2 are available. The production of GCA, GCB and GCC must be scheduled to at least cover the minimum demand level of 6,000, 7,000 and 9,000 l, respectively. It is assumed that when chemical components are mixed, there is no loss or gain in volume.

Each chemical component, C1 and C2, has a proportional critical element, 0.4 and 0.2, respectively. That is to say, each litre of C1 contains 0.4 l of the critical element. To obtain GCA, the mixture must proportionally contain at least a 0.3 fraction of the critical element. Another requirement is that the quantity of the

critical element is seen in GCB, an 0.3 fraction at the most.
Furthermore, the minimum ratio of C1 with C2 in product GCC must be 0.3.

The profit expected for the sale of each litre of GCA, GCB and GCC is \$120, \$135 and \$155, respectively.

Optimise the production planning of this firm.

```
!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 1.1: Create Variables for Mathematical Model

```
message11 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt11,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)
```

```
response_p11 = message11.content[0].text
```

```
# Print response
print(response_p11)
```

 Let's define the variables for this optimization problem:

```
Decision variables:
Let x1 = Quantity of GCA produced (in liters)
Let x2 = Quantity of GCB produced (in liters)
Let x3 = Quantity of GCC produced (in liters)

Let y1 = Quantity of component C1 used in GCA (in liters)
Let y2 = Quantity of component C2 used in GCA (in liters)
Let y3 = Quantity of component C1 used in GCB (in liters)
Let y4 = Quantity of component C2 used in GCB (in liters)
Let y5 = Quantity of component C1 used in GCC (in liters)
Let y6 = Quantity of component C2 used in GCC (in liters)
```


These are the main variables needed to formulate the objective function and constraints for this optimization problem.

✓ 1. Prompt 1.2: Create Objective for Mathematical Model

```
message12 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt12,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11)}
    ]
)
```

```
response_p12 = message12.content[0].text
```

```
# Print response
print(response_p12)
```

 The objective is to maximize the total profit from selling the three products GCA, GCB, and GCC. The profit per liter fo

```
Maximize: 120x1 + 135x2 + 155x3
```

```
Where:
x1 is the quantity of GCA produced in liters
x2 is the quantity of GCB produced in liters
```

x_3 is the quantity of GCC produced in liters

The coefficients 120, 135, and 155 represent the profit per liter for GCA, GCB, and GCC, respectively. By maximizing thi

✓ 1. Prompt 1.3: Create Constraints for Mathematical Model

```
message13 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt13,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11 + response_p12)}
    ]
)

response_p13 = message13.content[0].text

# Print response
print(response_p13)
```

↩ Here are the constraints for this optimization problem:

1. Component availability constraints:
 - $y_1 + y_3 + y_5 \leq 10000$ (C1 availability)
 - $y_2 + y_4 + y_6 \leq 15000$ (C2 availability)
2. Minimum demand constraints:
 - $x_1 \geq 6000$ (GCA minimum demand)
 - $x_2 \geq 7000$ (GCB minimum demand)
 - $x_3 \geq 9000$ (GCC minimum demand)
3. Product composition constraints:
 - $x_1 = y_1 + y_2$ (GCA composition)
 - $x_2 = y_3 + y_4$ (GCB composition)
 - $x_3 = y_5 + y_6$ (GCC composition)
4. Critical element constraints:
 - $0.4y_1 + 0.2y_2 \geq 0.3x_1$ (GCA critical element requirement)
 - $0.4y_3 + 0.2y_4 \leq 0.3x_2$ (GCB critical element requirement)
5. Component ratio constraint for GCC:
 - $y_5 \geq 0.3y_6$ (Minimum ratio of C1 to C2 in GCC)
6. Non-negativity constraints:
 - $x_1, x_2, x_3, y_1, y_2, y_3, y_4, y_5, y_6 \geq 0$

These constraints ensure that:

1. The available quantities of components C1 and C2 are not exceeded.
2. The minimum demand for each product is met.
3. The products are composed of the correct components.
4. The critical element requirements for GCA and GCB are satisfied.
5. The minimum ratio of C1 to C2 in GCC is maintained.
6. All decision variables are non-negative.

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

In case more than the component of each step was generated, use "last instance"/ "newest version" of the components.

Example: Prompt 1.1 (Variables): Model generates only variables Prompt 1.2 (Objective): Model generates objective and constraints Prompt 1.3 (Constraints): Model generates only constraints

Use Variables from step 1.1, use objective from 1.2 and use constraints from 1.3 (since constraints version 1.3 is more recent than constraints version 1.2).

This set up allows for potential self-correction of the model throughout the three prompts.

Correctness of each component is still generated in the respective step, so self-correction is independent of this.

```

message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": (response_p11 + response_p12 + response_p13)}
    ]
)

response_p2 = message2.content[0].text

# Print response
print(response_p2)

from pyomo.environ import *

# Create the model
model = ConcreteModel()

# Define the decision variables
model.x1 = Var(within=NonNegativeReals)
model.x2 = Var(within=NonNegativeReals)
model.x3 = Var(within=NonNegativeReals)
model.y1 = Var(within=NonNegativeReals)
model.y2 = Var(within=NonNegativeReals)
model.y3 = Var(within=NonNegativeReals)
model.y4 = Var(within=NonNegativeReals)
model.y5 = Var(within=NonNegativeReals)
model.y6 = Var(within=NonNegativeReals)

# Define the objective function
model.profit = Objective(expr=120*model.x1 + 135*model.x2 + 155*model.x3, sense=maximize)

# Define the constraints
model.c1_availability = Constraint(expr=model.y1 + model.y3 + model.y5 <= 10000)
model.c2_availability = Constraint(expr=model.y2 + model.y4 + model.y6 <= 15000)
model.gca_min_demand = Constraint(expr=model.x1 >= 6000)
model.gcb_min_demand = Constraint(expr=model.x2 >= 7000)
model.gcc_min_demand = Constraint(expr=model.x3 >= 9000)
model.gca_composition = Constraint(expr=model.x1 == model.y1 + model.y2)
model.gcb_composition = Constraint(expr=model.x2 == model.y3 + model.y4)
model.gcc_composition = Constraint(expr=model.x3 == model.y5 + model.y6)
model.gca_critical_element = Constraint(expr=0.4*model.y1 + 0.2*model.y2 >= 0.3*model.x1)
model.gcb_critical_element = Constraint(expr=0.4*model.y3 + 0.2*model.y4 <= 0.3*model.x2)
model.gcc_component_ratio = Constraint(expr=model.y5 >= 0.3*model.y6)

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

# Print the results
print("Optimal solution:")
print("Quantity of GCA produced (x1):", model.x1.value)
print("Quantity of GCB produced (x2):", model.x2.value)
print("Quantity of GCC produced (x3):", model.x3.value)
print("Quantity of C1 used in GCA (y1):", model.y1.value)
print("Quantity of C2 used in GCA (y2):", model.y2.value)
print("Quantity of C1 used in GCB (y3):", model.y3.value)
print("Quantity of C2 used in GCB (y4):", model.y4.value)
print("Quantity of C1 used in GCC (y5):", model.y5.value)
print("Quantity of C2 used in GCC (y6):", model.y6.value)
print("Total profit:", model.profit.expr())

```

In this code, we first create the Pyomo model and define the decision variables using the `Var` function with the `NonNegativeReals` domain. Next, we define the objective function using the `Objective` function, specifying the expression for the total profit. We then define the constraints using the `Constraint` function, with each constraint represented by its corresponding expression. Finally, we solve the optimization problem using the GLPK solver (assuming it is installed) and print the optimal solution.

4. Input Problem Data and Test Model Code

```

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

```

 [Show hidden output](#)

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

```

# Create the model
model = ConcreteModel()

# Define the decision variables
model.x1 = Var(within=NonNegativeReals)
model.x2 = Var(within=NonNegativeReals)
model.x3 = Var(within=NonNegativeReals)
model.y1 = Var(within=NonNegativeReals)
model.y2 = Var(within=NonNegativeReals)
model.y3 = Var(within=NonNegativeReals)
model.y4 = Var(within=NonNegativeReals)
model.y5 = Var(within=NonNegativeReals)
model.y6 = Var(within=NonNegativeReals)

# Define the objective function
model.profit = Objective(expr=120*model.x1 + 135*model.x2 + 155*model.x3, sense=maximize)

# Define the constraints
model.c1_availability = Constraint(expr=model.y1 + model.y3 + model.y5 <= 10000)
model.c2_availability = Constraint(expr=model.y2 + model.y4 + model.y6 <= 15000)
model.gca_min_demand = Constraint(expr=model.x1 >= 6000)
model.gcb_min_demand = Constraint(expr=model.x2 >= 7000)
model.gcc_min_demand = Constraint(expr=model.x3 >= 9000)
model.gca_composition = Constraint(expr=model.x1 == model.y1 + model.y2)
model.gcb_composition = Constraint(expr=model.x2 == model.y3 + model.y4)
model.gcc_composition = Constraint(expr=model.x3 == model.y5 + model.y6)
model.gca_critical_element = Constraint(expr=0.4*model.y1 + 0.2*model.y2 >= 0.3*model.x1)
model.gcb_critical_element = Constraint(expr=0.4*model.y3 + 0.2*model.y4 <= 0.3*model.x2)
model.gcc_component_ratio = Constraint(expr=model.y5 >= 0.3*model.y6)

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

# Print the results
print("Optimal solution:")
print("Quantity of GCA produced (x1):", model.x1.value)
print("Quantity of GCB produced (x2):", model.x2.value)
print("Quantity of GCC produced (x3):", model.x3.value)
print("Quantity of C1 used in GCA (y1):", model.y1.value)
print("Quantity of C2 used in GCA (y2):", model.y2.value)
print("Quantity of C1 used in GCB (y3):", model.y3.value)
print("Quantity of C2 used in GCB (y4):", model.y4.value)
print("Quantity of C1 used in GCC (y5):", model.y5.value)
print("Quantity of C2 used in GCC (y6):", model.y6.value)
print("Total profit:", model.profit.expr())

```

```

➡ Optimal solution:
Quantity of GCA produced (x1): 6000.0
Quantity of GCB produced (x2): 7000.0
Quantity of GCC produced (x3): 12000.0
Quantity of C1 used in GCA (y1): 3000.0
Quantity of C2 used in GCA (y2): 3000.0
Quantity of C1 used in GCB (y3): 0.0
Quantity of C2 used in GCB (y4): 7000.0
Quantity of C1 used in GCC (y5): 7000.0
Quantity of C2 used in GCC (y6): 5000.0
Total profit: 3525000.0

```

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