

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)
```

 Here are the variables for this mathematical optimization problem:

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

y11 = quantity of component C1 used in GCA (in liters)
y12 = quantity of component C2 used in GCA (in liters)
y21 = quantity of component C1 used in GCB (in liters)
y22 = quantity of component C2 used in GCB (in liters)
y31 = quantity of component C1 used in GCC (in liters)
y32 = quantity of component C2 used in GCC (in liters)
```

These are the key decision variables that need to be determined to optimize the production planning based on the given c

✓ 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)
```

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

```
maximize 120x1 + 135x2 + 155x3
```

```
where:
```

```
x1 = quantity of GCA produced (in liters)
```

x2 = quantity of GCB produced (in liters)
 x3 = quantity of GCC produced (in liters)

The objective function represents the total profit earned by producing and selling the three chemical products GCA, GCB

✓ 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 mathematical optimization problem:

1. Component availability constraints:
 $y_{11} + y_{21} + y_{31} \leq 10,000$ (C1 availability)
 $y_{12} + y_{22} + y_{32} \leq 15,000$ (C2 availability)
2. Demand constraints:
 $x_1 \geq 6,000$ (minimum demand for GCA)
 $x_2 \geq 7,000$ (minimum demand for GCB)
 $x_3 \geq 9,000$ (minimum demand for GCC)
3. Product composition constraints:
 $x_1 = y_{11} + y_{12}$ (composition of GCA)
 $x_2 = y_{21} + y_{22}$ (composition of GCB)
 $x_3 = y_{31} + y_{32}$ (composition of GCC)
4. Critical element constraints:
 $0.4y_{11} + 0.2y_{12} \geq 0.3x_1$ (minimum critical element in GCA)
 $0.4y_{21} + 0.2y_{22} \leq 0.3x_2$ (maximum critical element in GCB)
5. Component ratio constraint:
 $y_{31} \geq 0.3y_{32}$ (minimum ratio of C1 to C2 in GCC)
6. Non-negativity constraints:
 $x_1, x_2, x_3, y_{11}, y_{12}, y_{21}, y_{22}, y_{31}, y_{32} \geq 0$ (all variables must be non-negative)

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 variables are non-negative, as negative quantities are not possible in this context.

✓ 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)

```

Here's the Python code using Pyomo to solve the optimization problem:

```

```python
from pyomo.environ import *

Create a Pyomo model
model = ConcreteModel()

Define decision variables
model.x1 = Var(within=NonNegativeReals)
model.x2 = Var(within=NonNegativeReals)
model.x3 = Var(within=NonNegativeReals)
model.y11 = Var(within=NonNegativeReals)
model.y12 = Var(within=NonNegativeReals)
model.y21 = Var(within=NonNegativeReals)
model.y22 = Var(within=NonNegativeReals)
model.y31 = Var(within=NonNegativeReals)
model.y32 = Var(within=NonNegativeReals)

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

Define constraints
model.c1_availability = Constraint(expr=model.y11 + model.y21 + model.y31 <= 10000)
model.c2_availability = Constraint(expr=model.y12 + model.y22 + model.y32 <= 15000)
model.demand_gca = Constraint(expr=model.x1 >= 6000)
model.demand_gcb = Constraint(expr=model.x2 >= 7000)
model.demand_gcc = Constraint(expr=model.x3 >= 9000)
model.composition_gca = Constraint(expr=model.x1 == model.y11 + model.y12)
model.composition_gcb = Constraint(expr=model.x2 == model.y21 + model.y22)
model.composition_gcc = Constraint(expr=model.x3 == model.y31 + model.y32)
model.critical_element_gca = Constraint(expr=0.4*model.y11 + 0.2*model.y12 >= 0.3*model.x1)
model.critical_element_gcb = Constraint(expr=0.4*model.y21 + 0.2*model.y22 <= 0.3*model.x2)
model.component_ratio_gcc = Constraint(expr=model.y31 >= 0.3*model.y32)

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

Print the results
print("Optimization Results:")
print("-----")
print(f"Profit: ${model.profit():.2f}")
print(f"GCA Produced: {model.x1():.2f} liters")
print(f"GCB Produced: {model.x2():.2f} liters")
print(f"GCC Produced: {model.x3():.2f} liters")
print(f"C1 used in GCA: {model.y11():.2f} liters")
print(f"C2 used in GCA: {model.y12():.2f} liters")
print(f"C1 used in GCB: {model.y21():.2f} liters")
print(f"C2 used in GCB: {model.y22():.2f} liters")
print(f"C1 used in GCC: {model.y31():.2f} liters")
print(f"C2 used in GCC: {model.y32():.2f} liters")
```

```

Note: This code assumes you have Pyomo and the GLPK solver installed. You can install them using `pip install pyomo glpk`.

The code defines the decision variables, objective function, and constraints based on the given problem formulation. I

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 a Pyomo model
model = ConcreteModel()

# Define decision variables
model.x1 = Var(within=NonNegativeReals)
model.x2 = Var(within=NonNegativeReals)
model.x3 = Var(within=NonNegativeReals)
model.y11 = Var(within=NonNegativeReals)
model.y12 = Var(within=NonNegativeReals)
model.y21 = Var(within=NonNegativeReals)
model.y22 = Var(within=NonNegativeReals)
model.y31 = Var(within=NonNegativeReals)
model.y32 = Var(within=NonNegativeReals)

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

# Define constraints
model.c1_availability = Constraint(expr=model.y11 + model.y21 + model.y31 <= 10000)
model.c2_availability = Constraint(expr=model.y12 + model.y22 + model.y32 <= 15000)
model.demand_gca = Constraint(expr=model.x1 >= 6000)
model.demand_gcb = Constraint(expr=model.x2 >= 7000)
model.demand_gcc = Constraint(expr=model.x3 >= 9000)
model.composition_gca = Constraint(expr=model.x1 == model.y11 + model.y12)
model.composition_gcb = Constraint(expr=model.x2 == model.y21 + model.y22)
model.composition_gcc = Constraint(expr=model.x3 == model.y31 + model.y32)
model.critical_element_gca = Constraint(expr=0.4*model.y11 + 0.2*model.y12 >= 0.3*model.x1)
model.critical_element_gcb = Constraint(expr=0.4*model.y21 + 0.2*model.y22 <= 0.3*model.x2)
model.component_ratio_gcc = Constraint(expr=model.y31 >= 0.3*model.y32)

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

# Print the results
print("Optimization Results:")
print("-----")
print(f"Profit: ${model.profit():.2f}")
print(f"GCA Produced: {model.x1():.2f} liters")
print(f"GCB Produced: {model.x2():.2f} liters")
print(f"GCC Produced: {model.x3():.2f} liters")
print(f"C1 used in GCA: {model.y11():.2f} liters")
print(f"C2 used in GCA: {model.y12():.2f} liters")
print(f"C1 used in GCB: {model.y21():.2f} liters")
print(f"C2 used in GCB: {model.y22():.2f} liters")
print(f"C1 used in GCC: {model.y31():.2f} liters")
print(f"C2 used in GCC: {model.y32():.2f} liters")
```

↗ Optimization Results:

```
-----
Profit: $3525000.00
GCA Produced: 6000.00 liters
GCB Produced: 7000.00 liters
GCC Produced: 12000.00 liters
C1 used in GCA: 3000.00 liters
C2 used in GCA: 3000.00 liters
C1 used in GCB: 0.00 liters
C2 used in GCB: 7000.00 liters
C1 used in GCC: 7000.00 liters
C2 used in GCC: 5000.00 liters
```

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