

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)
```

 Great, let's start by defining the variables for this optimization problem.

```
Let:
- x1 be the number of litres of GCA produced
- x2 be the number of litres of GCB produced
- x3 be the number of litres of GCC produced
- a1 be the number of litres of C1 used in producing GCA
- a2 be the number of litres of C1 used in producing GCB
- a3 be the number of litres of C1 used in producing GCC
- b1 be the number of litres of C2 used in producing GCA
- b2 be the number of litres of C2 used in producing GCB
- b3 be the number of litres of C2 used in producing GCC
```


These are the decision variables that represent the quantities we need to determine to optimize the production planning.

✓ 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)
```

 Great! Now that we have defined the decision variables, let's formulate the objective function for this optimization problem.

The firm's goal is to maximize the total profit from selling the three products: GCA, GCB, and GCC. The profit per litre

Objective Function:

Maximize $Z = 120x_1 + 135x_2 + 155x_3$

This function represents the total profit earned by selling x_1 litres of GCA, x_2 litres of GCB, and x_3 litres of GCC. By

✓ 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)
```

➦ Now, let's formulate the constraints for this optimization problem:

1. Component availability constraints:
 - $a_1 + a_2 + a_3 \leq 10,000$ (C1 availability)
 - $b_1 + b_2 + b_3 \leq 15,000$ (C2 availability)
2. Minimum demand constraints:
 - $x_1 \geq 6,000$ (GCA minimum demand)
 - $x_2 \geq 7,000$ (GCB minimum demand)
 - $x_3 \geq 9,000$ (GCC minimum demand)
3. No loss or gain in volume constraints:
 - $a_1 + b_1 = x_1$ (GCA volume balance)
 - $a_2 + b_2 = x_2$ (GCB volume balance)
 - $a_3 + b_3 = x_3$ (GCC volume balance)
4. Critical element constraints:
 - $0.4a_1 + 0.2b_1 \geq 0.3x_1$ (GCA critical element requirement)
 - $0.4a_2 + 0.2b_2 \leq 0.3x_2$ (GCB critical element requirement)
5. Component ratio constraint for GCC:
 - $a_3 \geq 0.3b_3$ (Minimum ratio of C1 to C2 in GCC)
6. Non-negativity constraints:
 - $x_1, x_2, x_3, a_1, a_2, a_3, b_1, b_2, b_3 \geq 0$

These constraints ensure that the optimal solution satisfies the given requirements, such as component availability, min

✓ 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=4096,
    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 for the optimization problem:

```

python
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.a1 = Var(within=NonNegativeReals)
model.a2 = Var(within=NonNegativeReals)
model.a3 = Var(within=NonNegativeReals)
model.b1 = Var(within=NonNegativeReals)
model.b2 = Var(within=NonNegativeReals)
model.b3 = 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.a1 + model.a2 + model.a3 <= 10000)
model.c2_availability = Constraint(expr=model.b1 + model.b2 + model.b3 <= 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_volume_balance = Constraint(expr=model.a1 + model.b1 == model.x1)
model.gcb_volume_balance = Constraint(expr=model.a2 + model.b2 == model.x2)
model.gcc_volume_balance = Constraint(expr=model.a3 + model.b3 == model.x3)
model.gca_critical_element = Constraint(expr=0.4*model.a1 + 0.2*model.b1 >= 0.3*model.x1)
model.gcb_critical_element = Constraint(expr=0.4*model.a2 + 0.2*model.b2 <= 0.3*model.x2)
model.gcc_component_ratio = Constraint(expr=model.a3 >= 0.3*model.b3)

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

# Print the results
print("Optimal Production Plan:")
print("GCA (x1):", model.x1.value)
print("GCB (x2):", model.x2.value)
print("GCC (x3):", model.x3.value)
print("C1 used in GCA (a1):", model.a1.value)
print("C1 used in GCB (a2):", model.a2.value)
print("C1 used in GCC (a3):", model.a3.value)
print("C2 used in GCA (b1):", model.b1.value)
print("C2 used in GCB (b2):", model.b2.value)
print("C2 used in GCC (b3):", model.b3.value)
print("Total Profit: $", model.profit())

```

This code defines the optimization model, including the decision variables, objective function, and constraints. It th

Note that no sample data is used in this code, as the problem statement provides all the necessary information. The co

✓ 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.a1 = Var(within=NonNegativeReals)
model.a2 = Var(within=NonNegativeReals)
model.a3 = Var(within=NonNegativeReals)
model.b1 = Var(within=NonNegativeReals)
model.b2 = Var(within=NonNegativeReals)
model.b3 = Var(within=NonNegativeReals)

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

```

```
# Define the constraints
model.c1_availability = Constraint(expr=model.a1 + model.a2 + model.a3 <= 10000)
model.c2_availability = Constraint(expr=model.b1 + model.b2 + model.b3 <= 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_volume_balance = Constraint(expr=model.a1 + model.b1 == model.x1)
model.gcb_volume_balance = Constraint(expr=model.a2 + model.b2 == model.x2)
model.gcc_volume_balance = Constraint(expr=model.a3 + model.b3 == model.x3)
model.gca_critical_element = Constraint(expr=0.4*model.a1 + 0.2*model.b1 >= 0.3*model.x1)
model.gcb_critical_element = Constraint(expr=0.4*model.a2 + 0.2*model.b2 <= 0.3*model.x2)
model.gcc_component_ratio = Constraint(expr=model.a3 >= 0.3*model.b3)

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

# Print the results
print("Optimal Production Plan:")
print("GCA (x1):", model.x1.value)
print("GCB (x2):", model.x2.value)
print("GCC (x3):", model.x3.value)
print("C1 used in GCA (a1):", model.a1.value)
print("C1 used in GCB (a2):", model.a2.value)
print("C1 used in GCC (a3):", model.a3.value)
print("C2 used in GCA (b1):", model.b1.value)
print("C2 used in GCB (b2):", model.b2.value)
print("C2 used in GCC (b3):", model.b3.value)
print("Total Profit: $", model.profit())
```

```
➦ Optimal Production Plan:
GCA (x1): 6000.0
GCB (x2): 7000.0
GCC (x3): 12000.0
C1 used in GCA (a1): 3000.0
C1 used in GCB (a2): 0.0
C1 used in GCC (a3): 7000.0
C2 used in GCA (b1): 3000.0
C2 used in GCB (b2): 7000.0
C2 used in GCC (b3): 5000.0
Total Profit: $ 3525000.0
```

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