## 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/LP1.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:
 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: 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)
```

```
To optimize the production planning, we can formulate a linear programming model. Let's define the parameters, decision

Parameters:
- C1_max = 10,000 (maximum available quantity of C1 in litres)
- C2_max = 15,000 (maximum available quantity of C2 in litres)
- GCA_min = 6,000 (minimum demand for GCA in litres)
- GCB_min = 7,000 (minimum demand for GCB in litres)
- GCC_min = 9,000 (minimum demand for GCC in litres)
- C1_crit = 0.4 (critical element proportion in C1)
- C2_crit = 0.2 (critical element proportion in C2)
- GCA_crit_min = 0.3 (minimum critical element proportion in GCA)
- GCB_crit_max = 0.3 (maximum critical element proportion in GCB)
- GCC_ratio_min = 0.3 (minimum ratio of C1 to C2 in GCC)
- GCA_profit = 120 (profit per litre of GCA)
- GCB_profit = 135 (profit per litre of GCB)
- GCC_profit = 155 (profit per litre of GCC)

Decision Variables:
- x1: quantity of C1 used in GCA (in litres)
- x2: quantity of C2 used in GCA (in litres)
- y1: quantity of C1 used in GCB (in litres)
- y2: quantity of C2 used in GCB (in litres)
- z1: quantity of C1 used in GCC (in litres)
- z2: quantity of C2 used in GCC (in litres)

Objective Function:
Maximize Total Profit = GCA_profit * (x1 + x2) + GCB_profit * (y1 + y2) + GCC_profit * (z1 + z2)

Constraints:
1. Component Availability:
   - x1 + y1 + z1 ≤ C1_max
   - x2 + y2 + z2 ≤ C2_max
2. Minimum Demand:
   - x1 + x2 ≥ GCA_min
   - y1 + y2 ≥ GCB_min
   - z1 + z2 ≥ GCC_min
3. Critical Element Proportion:
   - C1_crit * x1 + C2_crit * x2 ≥ GCA_crit_min * (x1 + x2)
   - C1_crit * y1 + C2_crit * y2 ≤ GCB_crit_max * (y1 + y2)
4. Minimum Ratio in GCC:
   - z1 ≥ GCC_ratio_min * z2
5. Non-negativity:
   - x1, x2, y1, y2, z1, z2 ≥ 0

By solving this linear programming model, we can determine the optimal production quantities of GCA, GCB, and GCC that m
```

## 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.GCB_profit = Param(initialize=135)
model.GCC_profit = Param(initialize=155)

# Define the objective function
def obj_rule(model):
    return model.GCA_profit * (model.x1 + model.x2) + model.GCB_profit * (model.y1 + model.y2) + model.GCC_profit * (m
model.obj = Objective(rule=obj_rule, sense=maximize)

# Define the constraints
def component_availability_c1(model):
    return model.x1 + model.y1 + model.z1 <= model.C1_max
model.component_availability_c1_constraint = Constraint(rule=component_availability_c1)

def component_availability_c2(model):
    return model.x2 + model.y2 + model.z2 <= model.C2_max
model.component_availability_c2_constraint = Constraint(rule=component_availability_c2)

def minimum_demand_gca(model):
    return model.x1 + model.x2 >= model.GCA_min
model.minimum_demand_gca_constraint = Constraint(rule=minimum_demand_gca)

def minimum_demand_gcb(model):
    return model.y1 + model.y2 >= model.GCB_min
model.minimum_demand_gcb_constraint = Constraint(rule=minimum_demand_gcb)

def minimum_demand_gcc(model):
    return model.z1 + model.z2 >= model.GCC_min
model.minimum_demand_gcc_constraint = Constraint(rule=minimum_demand_gcc)

def critical_element_gca(model):
    return model.C1_crit * model.x1 + model.C2_crit * model.x2 >= model.GCA_crit_min * (model.x1 + model.x2)
model.critical_element_gca_constraint = Constraint(rule=critical_element_gca)

def critical_element_gcb(model):
    return model.C1_crit * model.y1 + model.C2_crit * model.y2 <= model.GCB_crit_max * (model.y1 + model.y2)
model.critical_element_gcb_constraint = Constraint(rule=critical_element_gcb)

def minimum_ratio_gcc(model):
    return model.z1 >= model.GCC_ratio_min * model.z2
model.minimum_ratio_gcc_constraint = Constraint(rule=minimum_ratio_gcc)

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

# Print the results
print("Optimal Solution:")
print("x1 =", model.x1.value)
print("x2 =", model.x2.value)
print("y1 =", model.y1.value)
print("y2 =", model.y2.value)
print("z1 =", model.z1.value)
print("z2 =", model.z2.value)
print("Total Profit =", model.obj())
```

In this code, I used sample data for the parameters as indicated in the problem description. You can replace these val
```

## 4. Input Problem Data and Test Model Code

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

Show hidden output
```

```
from pyomo.environ import *

# Create a Concrete Model
model = ConcreteModel()

# Define the decision variables
model.x1 = Var(within=NonNegativeReals)
model.x2 = Var(within=NonNegativeReals)
model.y1 = Var(within=NonNegativeReals)
model.y2 = Var(within=NonNegativeReals)
model.z1 = Var(within=NonNegativeReals)
model.z2 = Var(within=NonNegativeReals)

# Define the parameters (using sample data)
model.C1_max = Param(initialize=10000)
model.C2_max = Param(initialize=15000)
model.GCA_min = Param(initialize=6000)
model.GCB_min = Param(initialize=7000)
model.GCC_min = Param(initialize=9000)
model.C1_crit = Param(initialize=0.4)
model.C2_crit = Param(initialize=0.2)
model.GCA_crit_min = Param(initialize=0.3)
model.GCB_crit_max = Param(initialize=0.3)
model.GCC_ratio_min = Param(initialize=0.3)
model.GCA_profit = Param(initialize=120)
model.GCB_profit = Param(initialize=135)
model.GCC_profit = Param(initialize=155)

# Define the objective function
def obj_rule(model):
    return model.GCA_profit * (model.x1 + model.x2) + model.GCB_profit * (model.y1 + model.y2) + model.GCC_profit * (model.z
model.obj = Objective(rule=obj_rule, sense=maximize)

# Define the constraints
def component_availability_c1(model):
    return model.x1 + model.y1 + model.z1 <= model.C1_max
model.component_availability_c1_constraint = Constraint(rule=component_availability_c1)

def component_availability_c2(model):
    return model.x2 + model.y2 + model.z2 <= model.C2_max
model.component_availability_c2_constraint = Constraint(rule=component_availability_c2)

def minimum_demand_gca(model):
    return model.x1 + model.x2 >= model.GCA_min
model.minimum_demand_gca_constraint = Constraint(rule=minimum_demand_gca)

def minimum_demand_gcb(model):
    return model.y1 + model.y2 >= model.GCB_min
model.minimum_demand_gcb_constraint = Constraint(rule=minimum_demand_gcb)

def minimum_demand_gcc(model):
    return model.z1 + model.z2 >= model.GCC_min
model.minimum_demand_gcc_constraint = Constraint(rule=minimum_demand_gcc)

def critical_element_gca(model):
    return model.C1_crit * model.x1 + model.C2_crit * model.x2 >= model.GCA_crit_min * (model.x1 + model.x2)
model.critical_element_gca_constraint = Constraint(rule=critical_element_gca)

def critical_element_gcb(model):
    return model.C1_crit * model.y1 + model.C2_crit * model.y2 <= model.GCB_crit_max * (model.y1 + model.y2)
model.critical_element_gcb_constraint = Constraint(rule=critical_element_gcb)

def minimum_ratio_gcc(model):
    return model.z1 >= model.GCC_ratio_min * model.z2
model.minimum_ratio_gcc_constraint = Constraint(rule=minimum_ratio_gcc)

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

# Print the results
print("Optimal Solution:")
print("x1 =", model.x1.value)
print("x2 =", model.x2.value)
print("y1 =", model.y1.value)
print("y2 =", model.y2.value)
print("z1 =", model.z1.value)
print("z2 =", model.z2.value)
print("Total Profit =", model.obj())
```

```
Optimal Solution:
x1 = 3730.76923076923
x2 = 2269.23076923077
```

```
y1 = 3500.0
y2 = 3500.0
z1 = 2769.23076923077
z2 = 9230.76923076923
Total Profit = 3525000.0
```

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