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)
```

```
Let's formulate the mathematical optimization model for this problem.

    Parameters:
    - C1: Available quantity of component 1 (10,000 l)
    - C2: Available quantity of component 2 (15,000 l)
    - D_GCA, D_GCB, D_GCC: Minimum demand for products GCA, GCB, and GCC (6,000, 7,000, and 9,000 l, respectively)
    - CE_C1, CE_C2: Proportional critical element in components C1 and C2 (0.4 and 0.2, respectively)
    - CE_GCA_min: Minimum fraction of the critical element in product GCA (0.3)
    - CE_GCB_max: Maximum fraction of the critical element in product GCB (0.3)
    - R_GCC_min: Minimum ratio of C1 to C2 in product GCC (0.3)
    - P_GCA, P_GCB, P_GCC: Profit per litre for products GCA, GCB, and GCC ($120, $135, and $155, respectively)

    Decision Variables:
    - X_GCA_C1, X_GCA_C2: Quantity of components C1 and C2 used in product GCA
    - X_GCB_C1, X_GCB_C2: Quantity of components C1 and C2 used in product GCB
    - X_GCC_C1, X_GCC_C2: Quantity of components C1 and C2 used in product GCC

    Objective Function:
    Maximize Total Profit = P_GCA * (X_GCA_C1 + X_GCA_C2) + P_GCB * (X_GCB_C1 + X_GCB_C2) + P_GCC * (X_GCC_C1 + X_GCC_C2)

    Constraints:
    1. Component availability:
       - X_GCA_C1 + X_GCB_C1 + X_GCC_C1 ≤ C1
       - X_GCA_C2 + X_GCB_C2 + X_GCC_C2 ≤ C2

    2. Minimum demand:
       - X_GCA_C1 + X_GCA_C2 ≥ D_GCA
       - X_GCB_C1 + X_GCB_C2 ≥ D_GCB
       - X_GCC_C1 + X_GCC_C2 ≥ D_GCC

    3. Critical element requirements:
       - CE_C1 * X_GCA_C1 + CE_C2 * X_GCA_C2 ≥ CE_GCA_min * (X_GCA_C1 + X_GCA_C2)
       - CE_C1 * X_GCB_C1 + CE_C2 * X_GCB_C2 ≤ CE_GCB_max * (X_GCB_C1 + X_GCB_C2)

    4. Minimum ratio of C1 to C2 in GCC:
       - X_GCC_C1 ≥ R_GCC_min * X_GCC_C2

    5. Non-negativity:
       - X_GCA_C1, X_GCA_C2, X_GCB_C1, X_GCB_C2, X_GCC_C1, X_GCC_C2 ≥ 0

    The objective is to maximize the total profit by optimizing the production planning, subject to the given constraints on
```

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)
```

```
Great! Now let's implement the optimization model using Pyomo in Python.

    ```python
    from pyomo.environ import *

    # Create a Pyomo model
    model = ConcreteModel()

    # Define parameters
    model.C1 = 10000
    model.C2 = 15000
    model.D_GCA = 6000
    model.D_GCB = 7000
    model.D_GCC = 9000
    model.CE_C1 = 0.4
    model.CE_C2 = 0.2
    model.CE_GCA_min = 0.3
    model.CE_GCB_max = 0.3
    model.R_GCC_min = 0.3
    model.P_GCA = 120
    model.P_GCB = 135
    model.P_GCC = 155

    # Define decision variables
    model.X_GCA_C1 = Var(domain=NonNegativeReals)
    model.X_GCA_C2 = Var(domain=NonNegativeReals)
    model.X_GCB_C1 = Var(domain=NonNegativeReals)
    model.X_GCB_C2 = Var(domain=NonNegativeReals)
    model.X_GCC_C1 = Var(domain=NonNegativeReals)
    model.X_GCC_C2 = Var(domain=NonNegativeReals)

    # Define objective function
    def obj_rule(model):
        return model.P_GCA * (model.X_GCA_C1 + model.X_GCA_C2) + \
               model.P_GCB * (model.X_GCB_C1 + model.X_GCB_C2) + \
               model.P_GCC * (model.X_GCC_C1 + model.X_GCC_C2)

    model.OBJ = Objective(rule=obj_rule, sense=maximize)

    # Define constraints
    def component_availability_c1(model):
        return model.X_GCA_C1 + model.X_GCB_C1 + model.X_GCC_C1 <= model.C1

    def component_availability_c2(model):
        return model.X_GCA_C2 + model.X_GCB_C2 + model.X_GCC_C2 <= model.C2

    def minimum_demand_gca(model):
        return model.X_GCA_C1 + model.X_GCA_C2 >= model.D_GCA

    def minimum_demand_gcb(model):
        return model.X_GCB_C1 + model.X_GCB_C2 >= model.D_GCB

    def minimum_demand_gcc(model):
        return model.X_GCC_C1 + model.X_GCC_C2 >= model.D_GCC

    def critical_element_gca(model):
        return model.CE_C1 * model.X_GCA_C1 + model.CE_C2 * model.X_GCA_C2 >= model.CE_GCA_min * (model.X_GCA_C1 + model.X
```

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 Pyomo model
model = ConcreteModel()

# Define parameters
model.C1 = 10000
model.C2 = 15000
model.D_GCA = 6000
model.D_GCB = 7000
model.D_GCC = 9000
model.CE_C1 = 0.4
model.CE_C2 = 0.2
model.CE_GCA_min = 0.3
model.CE_GCB_max = 0.3
model.R_GCC_min = 0.3
model.P_GCA = 120
model.P_GCB = 135
model.P_GCC = 155

# Define decision variables
model.X_GCA_C1 = Var(domain=NonNegativeReals)
model.X_GCA_C2 = Var(domain=NonNegativeReals)
model.X_GCB_C1 = Var(domain=NonNegativeReals)
model.X_GCB_C2 = Var(domain=NonNegativeReals)
model.X_GCC_C1 = Var(domain=NonNegativeReals)
model.X_GCC_C2 = Var(domain=NonNegativeReals)

# Define objective function
def obj_rule(model):
    return model.P_GCA * (model.X_GCA_C1 + model.X_GCA_C2) + \
           model.P_GCB * (model.X_GCB_C1 + model.X_GCB_C2) + \
           model.P_GCC * (model.X_GCC_C1 + model.X_GCC_C2)

model.OBJ = Objective(rule=obj_rule, sense=maximize)

# Define constraints
def component_availability_c1(model):
    return model.X_GCA_C1 + model.X_GCB_C1 + model.X_GCC_C1 <= model.C1

def component_availability_c2(model):
    return model.X_GCA_C2 + model.X_GCB_C2 + model.X_GCC_C2 <= model.C2

def minimum_demand_gca(model):
    return model.X_GCA_C1 + model.X_GCA_C2 >= model.D_GCA

def minimum_demand_gcb(model):
    return model.X_GCB_C1 + model.X_GCB_C2 >= model.D_GCB

def minimum_demand_gcc(model):
    return model.X_GCC_C1 + model.X_GCC_C2 >= model.D_GCC

def critical_element_gca(model):
    return model.CE_C1 * model.X_GCA_C1 + model.CE_C2 * model.X_GCA_C2 >= model.CE_GCA_min * (model.X_GCA_C1 + model.X_GCA_C

def critical_element_gcb(model):
    return model.CE_C1 * model.X_GCB_C1 + model.CE_C2 * model.X_GCB_C2 <= model.CE_GCB_max * (model.X_GCB_C1 + model.X_GCB_C

def ratio_c1_c2_gcc(model):
    return model.X_GCC_C1 >= model.R_GCC_min * model.X_GCC_C2

model.C1_Availability = Constraint(rule=component_availability_c1)
model.C2_Availability = Constraint(rule=component_availability_c2)
model.GCA_MinDemand = Constraint(rule=minimum_demand_gca)
model.GCB_MinDemand = Constraint(rule=minimum_demand_gcb)
model.GCC_MinDemand = Constraint(rule=minimum_demand_gcc)
model.GCA_CriticalElement = Constraint(rule=critical_element_gca)
model.GCB_CriticalElement = Constraint(rule=critical_element_gcb)
model.GCC_RatioC1C2 = Constraint(rule=ratio_c1_c2_gcc)

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

# Print the optimal solution
print("Optimal Solution:")
print("X_GCA_C1 =", model.X_GCA_C1.value)
print("X_GCA_C2 =", model.X_GCA_C2.value)
print("X_GCB_C1 =", model.X_GCB_C1.value)
print("X_GCB_C2 =", model.X_GCB_C2.value)
print("X_GCC_C1 =", model.X_GCC_C1.value)
```

```
print("X_GCC_C2 =", model.X_GCC_C2.value)
print("Total Profit =", model.OBJ())
```

```
Optimal Solution:
X_GCA_C1 = 3730.76923076923
X_GCA_C2 = 2269.23076923077
X_GCB_C1 = 3500.0
X_GCB_C2 = 3500.0
X_GCC_C1 = 2769.23076923077
X_GCC_C2 = 9230.76923076923
Total Profit = 3525000.0
```

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