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 for the firm, we need to formulate a mathematical optimization model. Let's define t

    Parameters:
    - C1_avail = 10,000 (available quantity of component C1 in litres)
    - C2_avail = 15,000 (available quantity of component C2 in litres)
    - GCA_min = 6,000 (minimum demand for product GCA in litres)
    - GCB_min = 7,000 (minimum demand for product GCB in litres)
    - GCC_min = 9,000 (minimum demand for product GCC in litres)
    - C1_crit = 0.4 (critical element fraction in component C1)
    - C2_crit = 0.2 (critical element fraction in component C2)
    - GCA_crit_min = 0.3 (minimum critical element fraction in product GCA)
    - GCB_crit_max = 0.3 (maximum critical element fraction in product GCB)
    - GCC_ratio_min = 0.3 (minimum ratio of C1 to C2 in product GCC)
    - GCA_profit = 120 (profit per litre of product GCA)
    - GCB_profit = 135 (profit per litre of product GCB)
    - GCC_profit = 155 (profit per litre of product GCC)

    Decision Variables:
    - GCA_C1 (quantity of component C1 used in product GCA in litres)
    - GCA_C2 (quantity of component C2 used in product GCA in litres)
    - GCB_C1 (quantity of component C1 used in product GCB in litres)
    - GCB_C2 (quantity of component C2 used in product GCB in litres)
    - GCC_C1 (quantity of component C1 used in product GCC in litres)
    - GCC_C2 (quantity of component C2 used in product GCC in litres)

    Objective Function:
    Maximize Total Profit = GCA_profit * (GCA_C1 + GCA_C2) + GCB_profit * (GCB_C1 + GCB_C2) + GCC_profit * (GCC_C1 + GCC_C2)

    Constraints:
    1. Component Availability:
       - GCA_C1 + GCB_C1 + GCC_C1 ≤ C1_avail
       - GCA_C2 + GCB_C2 + GCC_C2 ≤ C2_avail

    2. Minimum Demand:
       - GCA_C1 + GCA_C2 ≥ GCA_min
       - GCB_C1 + GCB_C2 ≥ GCB_min
       - GCC_C1 + GCC_C2 ≥ GCC_min

    3. Critical Element Requirements:
       - (GCA_C1 * C1_crit + GCA_C2 * C2_crit) / (GCA_C1 + GCA_C2) ≥ GCA_crit_min
       - (GCB_C1 * C1_crit + GCB_C2 * C2_crit) / (GCB_C1 + GCB_C2) ≤ GCB_crit_max

    4. Minimum Component Ratio in GCC:
       - GCC_C1 / GCC_C2 ≥ GCC_ratio_min

    5. Non-negativity:
       - GCA_C1, GCA_C2, GCB_C1, GCB_C2, GCC_C1, GCC_C2 ≥ 0

    The objective is to maximize the total profit by determining the optimal quantities of components C1 and C2 used in each
```

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 this optimization problem using Python and Pyomo. Here's the code:

    ```python
    from pyomo.environ import *

    # Create a Pyomo model
    model = ConcreteModel()

    # Define decision variables
    model.GCA_C1 = Var(within=NonNegativeReals)
    model.GCA_C2 = Var(within=NonNegativeReals)
    model.GCB_C1 = Var(within=NonNegativeReals)
    model.GCB_C2 = Var(within=NonNegativeReals)
    model.GCC_C1 = Var(within=NonNegativeReals)
    model.GCC_C2 = Var(within=NonNegativeReals)

    # Define parameters
    model.C1_avail = Param(initialize=10000)
    model.C2_avail = 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 objective function
    def total_profit(model):
        return (
            model.GCA_profit * (model.GCA_C1 + model.GCA_C2) +
            model.GCB_profit * (model.GCB_C1 + model.GCB_C2) +
            model.GCC_profit * (model.GCC_C1 + model.GCC_C2)
        )
    model.profit = Objective(rule=total_profit, sense=maximize)

    # Define constraints
    def component_availability_C1(model):
        return model.GCA_C1 + model.GCB_C1 + model.GCC_C1 <= model.C1_avail
    model.comp_avail_C1 = Constraint(rule=component_availability_C1)

    def component_availability_C2(model):
        return model.GCA_C2 + model.GCB_C2 + model.GCC_C2 <= model.C2_avail
    model.comp_avail_C2 = Constraint(rule=component_availability_C2)

    def minimum_demand_GCA(model):
        return model.GCA_C1 + model.GCA_C2 >= model.GCA_min
    model.min_demand_GCA = Constraint(rule=minimum_demand_GCA)

    def minimum_demand_GCB(model):
        return model.GCB_C1 + model.GCB_C2 >= model.GCB_min
    model.min_demand_GCB = Constraint(rule=minimum_demand_GCB)

    def minimum_demand_GCC(model):
```

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 decision variables
model.GCA_C1 = Var(within=NonNegativeReals)
model.GCA_C2 = Var(within=NonNegativeReals)
model.GCB_C1 = Var(within=NonNegativeReals)
model.GCB_C2 = Var(within=NonNegativeReals)
model.GCC_C1 = Var(within=NonNegativeReals)
model.GCC_C2 = Var(within=NonNegativeReals)

# Define parameters
model.C1_avail = Param(initialize=10000)
model.C2_avail = 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 objective function
def total_profit(model):
    return (
        model.GCA_profit * (model.GCA_C1 + model.GCA_C2) +
        model.GCB_profit * (model.GCB_C1 + model.GCB_C2) +
        model.GCC_profit * (model.GCC_C1 + model.GCC_C2)
    )
model.profit = Objective(rule=total_profit, sense=maximize)

# Define constraints
def component_availability_C1(model):
    return model.GCA_C1 + model.GCB_C1 + model.GCC_C1 <= model.C1_avail
model.comp_avail_C1 = Constraint(rule=component_availability_C1)

def component_availability_C2(model):
    return model.GCA_C2 + model.GCB_C2 + model.GCC_C2 <= model.C2_avail
model.comp_avail_C2 = Constraint(rule=component_availability_C2)

def minimum_demand_GCA(model):
    return model.GCA_C1 + model.GCA_C2 >= model.GCA_min
model.min_demand_GCA = Constraint(rule=minimum_demand_GCA)

def minimum_demand_GCB(model):
    return model.GCB_C1 + model.GCB_C2 >= model.GCB_min
model.min_demand_GCB = Constraint(rule=minimum_demand_GCB)

def minimum_demand_GCC(model):
    return model.GCC_C1 + model.GCC_C2 >= model.GCC_min
model.min_demand_GCC = Constraint(rule=minimum_demand_GCC)

def critical_element_GCA(model):
    return (model.GCA_C1 * model.C1_crit + model.GCA_C2 * model.C2_crit) / (model.GCA_C1 + model.GCA_C2) >= model.GCA_crit_m
model.crit_elem_GCA = Constraint(rule=critical_element_GCA)

def critical_element_GCB(model):
    return (model.GCB_C1 * model.C1_crit + model.GCB_C2 * model.C2_crit) / (model.GCB_C1 + model.GCB_C2) <= model.GCB_crit_m
model.crit_elem_GCB = Constraint(rule=critical_element_GCB)

def minimum_ratio_GCC(model):
    return model.GCC_C1 / model.GCC_C2 >= model.GCC_ratio_min
model.min_ratio_GCC = Constraint(rule=minimum_ratio_GCC)

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

# Print the results
print("Optimal solution found!")
print("Objective value (Total Profit):", model.profit())
print("GCA_C1:", model.GCA_C1())
print("GCA_C2:", model.GCA_C2())
print("GCB_C1:", model.GCB_C1())
print("GCB_C2:", model.GCB_C2())
```

```
print("GCC_C1:", model.GCC_C1())
print("GCC_C2:", model.GCC_C2())
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-4859bf8f4108> in <cell line: 73>()
     71 # Solve the optimization problem
     72 solver = SolverFactory('glpk')
---> 73 results = solver.solve(model)
     74
     75 # Print the results

                                 9 frames
/usr/local/lib/python3.10/dist-packages/pyomo/repn/plugins/lp_writer.py in
write(self, model)
    422             repn = constraint_visitor.walk_expression(con.body)
    423             if repn.nonlinear is not None:
--> 424                 raise ValueError(
    425                     f"Model constraint ({con.name}) contains
nonlinear terms that "
    426                     "cannot be written to LP format"

ValueError: Model constraint (crit elem GCA) contains nonlinear terms that
```

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

```
# Installing pyomo nd solver
!pip install -q pyomo
!wget -N -q "https://matematica.unipv.it/gualandi/solvers/ipopt-linux64.zip"
!unzip -o -q ipopt-linux64
```

```
from pyomo.environ import *

# Create a Pyomo model
model = ConcreteModel()

# Define decision variables
model.GCA_C1 = Var(within=NonNegativeReals)
model.GCA_C2 = Var(within=NonNegativeReals)
model.GCB_C1 = Var(within=NonNegativeReals)
model.GCB_C2 = Var(within=NonNegativeReals)
model.GCC_C1 = Var(within=NonNegativeReals)
model.GCC_C2 = Var(within=NonNegativeReals)

# Define parameters
model.C1_avail = Param(initialize=10000)
model.C2_avail = 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 objective function
def total_profit(model):
    return (
        model.GCA_profit * (model.GCA_C1 + model.GCA_C2) +
        model.GCB_profit * (model.GCB_C1 + model.GCB_C2) +
        model.GCC_profit * (model.GCC_C1 + model.GCC_C2)
    )
model.profit = Objective(rule=total_profit, sense=maximize)

# Define constraints
def component_availability_C1(model):
    return model.GCA_C1 + model.GCB_C1 + model.GCC_C1 <= model.C1_avail
model.comp_avail_C1 = Constraint(rule=component_availability_C1)

def component_availability_C2(model):
    return model.GCA_C2 + model.GCB_C2 + model.GCC_C2 <= model.C2_avail
model.comp_avail_C2 = Constraint(rule=component_availability_C2)

def minimum_demand_GCA(model):
    return model.GCA_C1 + model.GCA_C2 >= model.GCA_min
model.min_demand_GCA = Constraint(rule=minimum_demand_GCA)
```