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/LP2.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:
 You are in charge of the supply purchasing of a company that produces two kinds of drugs.
The drugs contain a specific active agent, which is extracted from two different kinds of raw materials that should be p

The goal is to maximize the total profit obtained from producing the drugs, which means minimizing purchasing costs for

You are given a budget for purchasing raw materials and operating the production process of the drugs which cannot be ex
Additionally, you need to keep the capacity constraints for the production of the drugs in mind as there is only a limit
Your company can also only store a limited amount of raw materials.
Most importantly, the amount of active ingredient extracted from the raw materials you purchase needs to satisfy the req
Importantly, your experience with the production of the drugs has shown the the amount of active ingredient your process
During purchasing, you need to make sure that no matter how much the amount of active ingredient varies, the required am
```

```
!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)
```

```
Certainly! Here's a mathematical optimization model for the given problem:

Parameters:
- D: Set of drugs (indexed by i)
- R: Set of raw materials (indexed by j)
- p_i: Unit selling price of drug i
- c_j: Unit purchasing cost of raw material j
- o_i: Unit operational cost of producing drug i
- B: Budget for purchasing raw materials and operating the production process
- M: Available manpower hours
- E: Available equipment hours
- S_j: Storage capacity for raw material j
- a_ij: Amount of active ingredient extracted from one unit of raw material j for drug i
- r_i: Required amount of active ingredient in drug i
- α_j: Minimum percentage of active ingredient extracted from raw material j (α_1 = 0.995, α_2 = 0.98)

Decision Variables:
- x_i: Quantity of drug i to produce
- y_j: Quantity of raw material j to purchase

Objective Function:
Maximize total profit:
∑_i (p_i * x_i) - ∑_j (c_j * y_j) - ∑_i (o_i * x_i)

Constraints:
1. Budget constraint:
   ∑_j (c_j * y_j) + ∑_i (o_i * x_i) ≤ B

2. Manpower constraint:
   ∑_i (m_i * x_i) ≤ M, where m_i is the manpower hours required to produce one unit of drug i

3. Equipment constraint:
   ∑_i (e_i * x_i) ≤ E, where e_i is the equipment hours required to produce one unit of drug i

4. Storage constraint for each raw material j:
   y_j ≤ S_j, ∀j ∈ R

5. Active ingredient requirement for each drug i (considering the variation):
   ∑_j (α_j * a_ij * y_j) ≥ r_i * x_i, ∀i ∈ D

6. Non-negativity constraints:
   x_i ≥ 0, ∀i ∈ D
   y_j ≥ 0, ∀j ∈ R

This optimization model aims to maximize the total profit by determining the optimal quantities of drugs to produce (x_i
```

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)
```

```
    # Define objective function
    model.profit = pyo.Objective(
        expr=sum(model.p[i] * model.x[i] for i in model.D) -
             sum(model.c[j] * model.y[j] for j in model.R) -
             sum(model.o[i] * model.x[i] for i in model.D),
        sense=pyo.maximize
    )

    # Define constraints
    model.budget_constraint = pyo.Constraint(
        expr=sum(model.c[j] * model.y[j] for j in model.R) +
             sum(model.o[i] * model.x[i] for i in model.D) <= model.B
    )

    model.manpower_constraint = pyo.Constraint(
        expr=sum(1.5 * model.x[i] for i in model.D) <= model.M  # Sample data: 1.5 manpower hours per unit of drug
    )

    model.equipment_constraint = pyo.Constraint(
        expr=sum(2 * model.x[i] for i in model.D) <= model.E  # Sample data: 2 equipment hours per unit of drug
    )

    model.storage_constraints = pyo.ConstraintList()
    for j in model.R:
        model.storage_constraints.add(model.y[j] <= model.S[j])

    model.active_ingredient_constraints = pyo.ConstraintList()
    for i in model.D:
        model.active_ingredient_constraints.add(
            sum(model.alpha[j] * model.a[i, j] * model.y[j] for j in model.R) >= model.r[i] * model.x[i]
        )

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

    # Print the results
    print("Optimal Solution:")
    print("Objective Value (Total Profit):", pyo.value(model.profit))
    print("Drug Production Quantities:")
    for i in model.D:
        print(f"- {i}: {model.x[i].value}")
    print("Raw Material Purchase Quantities:")
    for j in model.R:
        print(f"- {j}: {model.y[j].value}")
    ```

    In this code, I used sample data for the manpower hours required per unit of drug (`1.5`) and the equipment hours requ

    The code defines the sets, parameters, decision variables, objective function, and constraints according to the provid
```

## 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

```
import pyomo.environ as pyo

# Define the model
model = pyo.ConcreteModel()

# Define sets
model.D = pyo.Set(initialize=['D1', 'D2'])  # Set of drugs
model.R = pyo.Set(initialize=['R1', 'R2'])  # Set of raw materials

# Define parameters
model.p = pyo.Param(model.D, initialize={'D1': 6200, 'D2': 6900})  # Unit selling price of drugs
model.c = pyo.Param(model.R, initialize={'R1': 100, 'R2': 199.90})  # Unit purchasing cost of raw materials
model.o = pyo.Param(model.D, initialize={'D1': 700, 'D2': 800})  # Unit operational cost of producing drugs
model.B = pyo.Param(initialize=100000)  # Budget
model.M = pyo.Param(initialize=2000)  # Available manpower hours
model.E = pyo.Param(initialize=800)  # Available equipment hours
model.S = pyo.Param(model.R, initialize={'R1': 1000, 'R2': 1000})  # Storage capacity for raw materials
model.a = pyo.Param(model.D, model.R, initialize={
    ('D1', 'R1'): 0.01, ('D1', 'R2'): 0.02,
    ('D2', 'R1'): 0.01, ('D2', 'R2'): 0.02
})  # Amount of active ingredient extracted from raw materials
model.r = pyo.Param(model.D, initialize={'D1': 0.5, 'D2': 0.6})  # Required amount of active ingredient in drugs
```
```

```
model.alpha = pyo.Param(model.R, initialize={'R1': 0.995, 'R2': 0.98})  # Minimum percentage of active ingredient extracted

# Define decision variables
model.x = pyo.Var(model.D, domain=pyo.NonNegativeReals)  # Quantity of drugs to produce
model.y = pyo.Var(model.R, domain=pyo.NonNegativeReals)  # Quantity of raw materials to purchase

# Define objective function
model.profit = pyo.Objective(
    expr=sum(model.p[i] * model.x[i] for i in model.D) -
         sum(model.c[j] * model.y[j] for j in model.R) -
         sum(model.o[i] * model.x[i] for i in model.D),
    sense=pyo.maximize
)

# Define constraints
model.budget_constraint = pyo.Constraint(
    expr=sum(model.c[j] * model.y[j] for j in model.R) +
         sum(model.o[i] * model.x[i] for i in model.D) <= model.B
)

model.manpower_constraint = pyo.Constraint(
    expr=sum(1.5 * model.x[i] for i in model.D) <= model.M  # Sample data: 1.5 manpower hours per unit of drug
)

model.equipment_constraint = pyo.Constraint(
    expr=sum(2 * model.x[i] for i in model.D) <= model.E  # Sample data: 2 equipment hours per unit of drug
)

model.storage_constraints = pyo.ConstraintList()
for j in model.R:
    model.storage_constraints.add(model.y[j] <= model.S[j])

model.active_ingredient_constraints = pyo.ConstraintList()
for i in model.D:
    model.active_ingredient_constraints.add(
        sum(model.alpha[j] * model.a[i, j] * model.y[j] for j in model.R) >= model.r[i] * model.x[i]
    )

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

# Print the results
print("Optimal Solution:")
print("Objective Value (Total Profit):", pyo.value(model.profit))
print("Drug Production Quantities:")
for i in model.D:
    print(f"- {i}: {model.x[i].value}")
print("Raw Material Purchase Quantities:")
for j in model.R:
    print(f"- {j}: {model.y[j].value}")
```

```
Optimal Solution:
Objective Value (Total Profit): 86958.51568437305
Drug Production Quantities:
- D1: 15.6450640740061
- D2: 13.0375533950051
Raw Material Purchase Quantities:
- R1: 786.184124321916
- R2: 0.0
```

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