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)
```

```
Thank you for providing the problem description. I will formulate a mathematical optimization model to address the suppl

Parameters:
- i: index for raw materials (i = 1, 2)
- j: index for drugs (j = 1, 2)
- p_i: purchasing cost per unit of raw material i
- c_j: operational cost per unit of drug j
- s_j: selling price per unit of drug j
- B: total budget for purchasing raw materials and operating the production process
- M_j: maximum production capacity for drug j (in units)
- R_i: maximum storage capacity for raw material i (in units)
- a_ij: amount of active ingredient extracted from one unit of raw material i for drug j
- d_j: required amount of active ingredient per unit of drug j
- v_i: maximum variation in the amount of active ingredient extracted from raw material i (0.5% for i=1, 2% for i=2)

Decision Variables:
- x_i: amount of raw material i to purchase (in units)
- y_j: amount of drug j to produce (in units)

Objective Function:
Maximize total profit:
∑(j=1 to 2) s_j * y_j - ∑(i=1 to 2) p_i * x_i - ∑(j=1 to 2) c_j * y_j

Constraints:
1. Budget constraint:
   ∑(i=1 to 2) p_i * x_i + ∑(j=1 to 2) c_j * y_j ≤ B

2. Production capacity constraints:
   y_j ≤ M_j, for j = 1, 2

3. Raw material storage constraints:
   x_i ≤ R_i, for i = 1, 2

4. Active ingredient requirement constraints (considering the maximum variation):
   ∑(i=1 to 2) a_ij * (1 - v_i) * x_i ≥ d_j * y_j, for j = 1, 2

5. Non-negativity constraints:
   x_i ≥ 0, for i = 1, 2
   y_j ≥ 0, for j = 1, 2

This mathematical optimization model aims to maximize the total profit by determining the optimal amounts of raw materia
```

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.I = RangeSet(1, 2)  # Set of raw materials
```

~~~
model.a = Param(model.I, model.J, initialize={(1, 1): 0.8, (1, 2): 0.6, (2, 1): 0.7, (2, 2): 0.9})  # Amount of active
model.d = Param(model.J, initialize={1: 0.5, 2: 0.4})  # Required amount of active ingredient per unit of drug
model.v = Param(model.I, initialize={1: 0.005, 2: 0.02})  # Maximum variation in the amount of active ingredient extra

# Define decision variables
model.x = Var(model.I, within=NonNegativeReals)  # Amount of raw material to purchase
model.y = Var(model.J, within=NonNegativeReals)  # Amount of drug to produce

# Define objective function
def profit_rule(model):
    return sum(model.s[j] * model.y[j] for j in model.J) - sum(model.p[i] * model.x[i] for i in model.I) - sum(model.c
model.profit = Objective(rule=profit_rule, sense=maximize)

# Define constraints
def budget_rule(model):
    return sum(model.p[i] * model.x[i] for i in model.I) + sum(model.c[j] * model.y[j] for j in model.J) <= model.B
model.budget_constraint = Constraint(rule=budget_rule)

def production_capacity_rule(model, j):
    return model.y[j] <= model.M[j]
model.production_capacity_constraint = Constraint(model.J, rule=production_capacity_rule)

def storage_capacity_rule(model, i):
    return model.x[i] <= model.R[i]
model.storage_capacity_constraint = Constraint(model.I, rule=storage_capacity_rule)

def active_ingredient_rule(model, j):
    return sum(model.a[i, j] * (1 - model.v[i]) * model.x[i] for i in model.I) >= model.d[j] * model.y[j]
model.active_ingredient_constraint = Constraint(model.J, rule=active_ingredient_rule)

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

# Print the results
print("Optimal solution found with total profit: ", model.profit())
print("Raw material purchases:")
for i in model.I:
    print(f"Raw material {i}: {model.x[i]():.2f} units")
print("Drug production:")
for j in model.J:
    print(f"Drug {j}: {model.y[j]():.2f} units")
```
~~~

In this code, I used sample data for the parameters, which are marked with comments. You can replace these sample valu

The code follows the mathematical optimization model described earlier, defining the sets, parameters, decision variab

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 sets
model.I = RangeSet(1, 2)  # Set of raw materials
model.J = RangeSet(1, 2)  # Set of drugs

# Define parameters (using sample data)
model.p = Param(model.I, initialize={1: 100, 2: 199.90})  # Purchasing cost per unit of raw material
model.c = Param(model.J, initialize={1: 700, 2: 800})  # Operational cost per unit of drug
model.s = Param(model.J, initialize={1: 6200, 2: 6900})  # Selling price per unit of drug
model.B = Param(initialize=100000)  # Total budget
model.M = Param(model.J, initialize={1: 100, 2: 80})  # Maximum production capacity for each drug
model.R = Param(model.I, initialize={1: 1000, 2: 1000})  # Maximum storage capacity for each raw material
model.a = Param(model.I, model.J, initialize={(1, 1): 0.01, (1, 2): 0.01, (2, 1): 0.02, (2, 2): 0.02})  # Amount of active in
model.d = Param(model.J, initialize={1: 0.5, 2: 0.6})  # Required amount of active ingredient per unit of drug
model.v = Param(model.I, initialize={1: 0.005, 2: 0.02})  # Maximum variation in the amount of active ingredient extracted

# Define decision variables
model.x = Var(model.I, within=NonNegativeReals)  # Amount of raw material to purchase
model.y = Var(model.J, within=NonNegativeReals)  # Amount of drug to produce

# Define objective function
def profit_rule(model):
```

```
def profit_rule(model):
    return sum(model.s[j] * model.y[j] for j in model.J) - sum(model.p[i] * model.x[i] for i in model.I) - sum(model.c[j] * m
model.profit = Objective(rule=profit_rule, sense=maximize)

# Define constraints
def budget_rule(model):
    return sum(model.p[i] * model.x[i] for i in model.I) + sum(model.c[j] * model.y[j] for j in model.J) <= model.B
model.budget_constraint = Constraint(rule=budget_rule)

def production_capacity_rule(model, j):
    return model.y[j] <= model.M[j]
model.production_capacity_constraint = Constraint(model.J, rule=production_capacity_rule)

def storage_capacity_rule(model, i):
    return model.x[i] <= model.R[i]
model.storage_capacity_constraint = Constraint(model.I, rule=storage_capacity_rule)

def active_ingredient_rule(model, j):
    return sum(model.a[i, j] * (1 - model.v[i]) * model.x[i] for i in model.I) >= model.d[j] * model.y[j]
model.active_ingredient_constraint = Constraint(model.J, rule=active_ingredient_rule)

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

# Print the results
print("Optimal solution found with total profit: ", model.profit())
print("Raw material purchases:")
for i in model.I:
    print(f"Raw material {i}: {model.x[i]():.2f} units")
print("Drug production:")
for j in model.J:
    print(f"Drug {j}: {model.y[j]():.2f} units")
```

```
Optimal solution found with total profit:  86958.51568437305
Raw material purchases:
Raw material 1: 786.18 units
Raw material 2: 0.00 units
Drug production:
Drug 1: 15.65 units
Drug 2: 13.04 units
```

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