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)
```

```
Here is a mathematical optimization model for the drug production problem:

Parameters:
- D: Set of drugs (index d)
- R: Set of raw materials (index r)
- p_d: Unit selling price of drug d
- c_r: Unit purchasing cost of raw material r
- o_d: Unit operational cost of producing drug d
- B: Budget for purchasing raw materials and operating production
- M_d: Maximum production capacity of drug d (limited by manpower and equipment hours)
- S_r: Maximum storage capacity for raw material r
- a_dr: Amount of active ingredient extracted from one unit of raw material r required for one unit of drug d
- α_r: Maximum percentage of variation in active ingredient extraction from raw material r (0.5% for r=1, 2% for r=2)

Decision Variables:
- x_d: Amount of drug d to produce
- y_r: Amount of raw material r to purchase

Objective Function:
Maximize total profit = total revenue - total purchasing cost - total operational cost
max ∑_{d ∈ D} p_d * x_d - ∑_{r ∈ R} c_r * y_r - ∑_{d ∈ D} o_d * x_d

Constraints:
1. Budget constraint:
   ∑_{r ∈ R} c_r * y_r + ∑_{d ∈ D} o_d * x_d ≤ B

2. Production capacity constraints for each drug d:
   x_d ≤ M_d, ∀d ∈ D

3. Storage capacity constraints for each raw material r:
   y_r ≤ S_r, ∀r ∈ R

4. Active ingredient requirement constraints for each drug d:
   ∑_{r ∈ R} (1 - α_r) * a_dr * y_r ≥ ∑_{d ∈ D} a_dr * x_d, ∀d ∈ D

5. Non-negativity constraints:
   x_d ≥ 0, ∀d ∈ D
   y_r ≥ 0, ∀r ∈ R

This optimization model aims to maximize the total profit by determining the optimal amount of drugs to produce (x_d) an
```

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)
```

~~~
Here's a Python Pyomo code for the drug production optimization problem:

```python
from pyomo.environ import *

# Define the model
model = ConcreteModel()

# Define sets
model.D = Set(initialize=['Drug1', 'Drug2'])  # Set of drugs (sample data)
model.R = Set(initialize=['Raw1', 'Raw2'])  # Set of raw materials (sample data)
~~~

```
    # Define parameters
    model.p = Param(model.D, initialize={'Drug1': 10, 'Drug2': 15})  # Unit selling price of drugs (sample data)
    model.c = Param(model.R, initialize={'Raw1': 5, 'Raw2': 8})  # Unit purchasing cost of raw materials (sample data)
    model.o = Param(model.D, initialize={'Drug1': 2, 'Drug2': 3})  # Unit operational cost of drugs (sample data)
    model.B = Param(initialize=1000)  # Budget (sample data)
    model.M = Param(model.D, initialize={'Drug1': 100, 'Drug2': 80})  # Maximum production capacity of drugs (sample data)
    model.S = Param(model.R, initialize={'Raw1': 500, 'Raw2': 400})  # Maximum storage capacity of raw materials (sample d
    model.a = Param(model.D, model.R, initialize={
        ('Drug1', 'Raw1'): 2, ('Drug1', 'Raw2'): 1,
        ('Drug2', 'Raw1'): 1, ('Drug2', 'Raw2'): 3
    })  # Amount of active ingredient required per unit of drug (sample data)
    model.alpha = Param(model.R, initialize={'Raw1': 0.005, 'Raw2': 0.02})  # Maximum percentage variation in extraction (

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

    # Define objective function
    def obj_rule(model):
        return sum(model.p[d] * model.x[d] for d in model.D) - \
               sum(model.c[r] * model.y[r] for r in model.R) - \
               sum(model.o[d] * model.x[d] for d in model.D)
    model.OBJ = Objective(rule=obj_rule, sense=maximize)

    # Define constraints
    def budget_constraint(model):
        return sum(model.c[r] * model.y[r] for r in model.R) + \
               sum(model.o[d] * model.x[d] for d in model.D) <= model.B
    model.budget_constraint = Constraint(rule=budget_constraint)

    def production_capacity_constraint(model, d):
        return model.x[d] <= model.M[d]
    model.production_capacity_constraint = Constraint(model.D, rule=production_capacity_constraint)

    def storage_capacity_constraint(model, r):
        return model.y[r] <= model.S[r]
    model.storage_capacity_constraint = Constraint(model.R, rule=storage_capacity_constraint)

    def active_ingredient_constraint(model, d):
        return sum((1 - model.alpha[r]) * model.a[d, r] * model.y[r] for r in model.R) >= \
               sum(model.a[d, r] * model.x[d] for r in model.R)
    model.active_ingredient_constraint = Constraint(model.D, rule=active_ingredient_constraint)

    # Solve the model
    solver = SolverFactory('glpk')
```

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 *

# Define the model
model = ConcreteModel()

# Define sets
model.D = Set(initialize=['Drug1', 'Drug2'])  # Set of drugs (sample data)
model.R = Set(initialize=['Raw1', 'Raw2'])  # Set of raw materials (sample data)

# Define parameters
model.p = Param(model.D, initialize={'Drug1': 6200, 'Drug2': 6900})  # Unit selling price of drugs (sample data)
model.c = Param(model.R, initialize={'Raw1': 100, 'Raw2': 199.90})  # Unit purchasing cost of raw materials (sample data)
model.o = Param(model.D, initialize={'Drug1': 700, 'Drug2': 800})  # Unit operational cost of drugs (sample data)
model.B = Param(initialize=100000)  # Budget (sample data)
model.M = Param(model.D, initialize={'Drug1': 100, 'Drug2': 80})  # Maximum production capacity of drugs (sample data)
model.S = Param(model.R, initialize={'Raw1': 1000, 'Raw2': 1000})  # Maximum storage capacity of raw materials (sample data)
model.a = Param(model.D, model.R, initialize={
    ('Drug1', 'Raw1'): 0.01, ('Drug1', 'Raw2'): 0.02,
    ('Drug2', 'Raw1'): 0.01, ('Drug2', 'Raw2'): 0.02
})  # Amount of active ingredient required per unit of drug (sample data)
model.alpha = Param(model.R, initialize={'Raw1': 0.005, 'Raw2': 0.02})  # Maximum percentage variation in extraction (sample

# Define decision variables
model.x = Var(model.D, within=NonNegativeReals)  # Amount of drugs to produce
model.y = Var(model.R, within=NonNegativeReals)  # Amount of raw materials to purchase
```

```
# Define objective function
def obj_rule(model):
    return sum(model.p[d] * model.x[d] for d in model.D) - \
           sum(model.c[r] * model.y[r] for r in model.R) - \
           sum(model.o[d] * model.x[d] for d in model.D)
model.OBJ = Objective(rule=obj_rule, sense=maximize)

# Define constraints
def budget_constraint(model):
    return sum(model.c[r] * model.y[r] for r in model.R) + \
           sum(model.o[d] * model.x[d] for d in model.D) <= model.B
model.budget_constraint = Constraint(rule=budget_constraint)

def production_capacity_constraint(model, d):
    return model.x[d] <= model.M[d]
model.production_capacity_constraint = Constraint(model.D, rule=production_capacity_constraint)

def storage_capacity_constraint(model, r):
    return model.y[r] <= model.S[r]
model.storage_capacity_constraint = Constraint(model.R, rule=storage_capacity_constraint)

def active_ingredient_constraint(model, d):
    return sum((1 - model.alpha[r]) * model.a[d, r] * model.y[r] for r in model.R) >= \
           sum(model.a[d, r] * model.x[d] for r in model.R)
model.active_ingredient_constraint = Constraint(model.D, rule=active_ingredient_constraint)

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

# Print the results
print("Optimal Profit: ", model.OBJ())
print("Drug Production:")
for d in model.D:
    print(f"{d}: {model.x[d]()}")
print("Raw Material Purchase:")
for r in model.R:
    print(f"{r}: {model.y[r]()}")
```

```
Optimal Profit:  627168.7587168763
Drug Production:
Drug1: 55.5090655509066
Drug2: 55.5090655509066
Raw Material Purchase:
Raw1: 167.364016736402
Raw2: 0.0
```

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