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/NL/NL3.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 buyer needs to acquire 239,600,480 units of a product and is considering bids from five suppliers, labeled A through
Each vendor has proposed different pricing structures, incorporating both setup fees and variable unit costs that change

The buyer's objective is to allocate the order among these suppliers to minimize overall costs, accounting for both setu

Vendor A offers a set up cost of $3855.34 and a unit cost of $61.150 per thousand of units.
Vendor A can supply up to 33 million units.

Vendor B offers a set up cost of $125,804.84 if purchasing between 22,000,000-70,000,000 units from vendor B with a unit
If purchasing between 70,000,001-100,000,000 units from vendor B, the set up cost increases to $269304.84 and the unit c
If purchasing between 100,000,001-150,000,000 units from vendor B, the unit cost per thousand units further decreases to
If purchasing between 150,000,001 and 160,000,000 units from vendor B, the unit cost is $62.119 per thousand units and t

Vendor C offers set up costs of $13,456.00 and a unit cost of $62.019 per thousand units.
Vendor C can supply up to 165.6 million units. Vendor D offers set up costs of $6,583.98 and a unit cost of $72.488 for

Vendor D can supply up to 12 million units at a price of $72.488 per thousand units and with a set up cost of $6583.98.

Vendor E offers free set up if purchasing between 0 and 42 million units of vendor E with a unit price of $70.150 per th
If purchasing between 42,000,001 and 77 million units from vendor E, the unit cost starts at $68.150 per thousand units,

Note that zero units may be purchased from vendor B: otherwise no positive number of units less than 22,000,000 may be p
```

```
!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"),
```

```
    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! Let's formulate this problem as a mathematical optimization model. We'll define the parameters, decision vari

Parameters:
- Q_total: Total quantity required (239,600,480 units)
- S_A, S_C, S_D: Setup costs for vendors A, C, and D
- S_B1, S_B2, S_B3, S_B4: Setup costs for vendor B based on quantity ranges
- S_E1, S_E2: Setup costs for vendor E based on quantity ranges
- C_A, C_C, C_D: Unit costs per thousand units for vendors A, C, and D
- C_B1, C_B2, C_B3, C_B4: Unit costs per thousand units for vendor B based on quantity ranges
- C_E1, C_E2: Unit costs per thousand units for vendor E based on quantity ranges
- R_E: Rate of price decrease for vendor E per million units purchased (0.05%)

Decision Variables:
- x_A, x_C, x_D: Quantity of units to be purchased from vendors A, C, and D
- x_B1, x_B2, x_B3, x_B4: Quantity of units to be purchased from vendor B for each quantity range
- x_E1, x_E2: Quantity of units to be purchased from vendor E for each quantity range
- y_B1, y_B2, y_B3, y_B4: Binary variables indicating if vendor B is selected for each quantity range
- y_E1, y_E2: Binary variables indicating if vendor E is selected for each quantity range

Objective Function:
Minimize: S_A + C_A * x_A / 1000 +
          S_B1 * y_B1 + C_B1 * x_B1 / 1000 +
          S_B2 * y_B2 + C_B2 * x_B2 / 1000 +
          S_B3 * y_B3 + C_B3 * x_B3 / 1000 +
          S_B4 * y_B4 + C_B4 * x_B4 / 1000 +
          S_C + C_C * x_C / 1000 +
          S_D + C_D * x_D / 1000 +
          S_E1 * y_E1 + C_E1 * x_E1 / 1000 +
          S_E2 * y_E2 + (C_E2 - R_E * (x_E2 / 1000000 - 42)) * x_E2 / 1000

Constraints:
1. Total quantity constraint: x_A + x_B1 + x_B2 + x_B3 + x_B4 + x_C + x_D + x_E1 + x_E2 = Q_total
2. Vendor capacity constraints:
   - x_A <= 33,000,000
   - x_B1 <= 70,000,000 * y_B1
   - x_B2 <= 100,000,000 * y_B2
   - x_B3 <= 150,000,000 * y_B3
   - x_B4 <= 160,000,000 * y_B4
   - x_C <= 165,600,000
   - x_D <= 12,000,000
   - x_E1 <= 42,000,000 * y_E1
   - x_E2 <= 77,000,000 * y_E2
3. Vendor B quantity range constraints:
   - 22,000,000 * y_B1 <= x_B1
   - 70,000,001 * y_B2 <= x_B2
   - 100,000,001 * y_B3 <= x_B3
   - 150,000,001 * y_B4 <= x_B4
4. Vendor B selection constraint: y_B1 + y_B2 + y_B3 + y_B4 <= 1
5. Vendor E quantity range constraints:
   - x_E1 <= 42,000,000 * y_E1
   - 42,000,001 * y_E2 <= x_E2
6. Vendor E selection constraint: y_E1 + y_E2 <= 1
7. Non-negativity constraints: All decision variables >= 0
8. Binary constraints: y_B1, y_B2, y_B3, y_B4, y_E1, y_E2 are binary variables

This optimization model minimizes the total cost, considering setup costs and unit costs for each vendor, while satisfyi
```

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 the Pyomo code for the optimization problem:

```python
from pyomo.environ import *

# Create the model
model = ConcreteModel()

# Define the parameters
model.Q_total = 239600480
model.S_A = 8000
model.S_C = 9500
model.S_D = 7200
model.S_B1, model.S_B2, model.S_B3, model.S_B4 = 5200, 7400, 8100, 10500
model.S_E1, model.S_E2 = 4900, 6100
model.C_A, model.C_C, model.C_D = 21.5, 19.8, 22.1
model.C_B1, model.C_B2, model.C_B3, model.C_B4 = 20.4, 19.8, 19.1, 18.6
model.C_E1, model.C_E2 = 23.1, 22.5
model.R_E = 0.05 / 100  # 0.05%

# Define the decision variables
model.x_A = Var(domain=NonNegativeReals)
model.x_C = Var(domain=NonNegativeReals)
model.x_D = Var(domain=NonNegativeReals)
model.x_B1 = Var(domain=NonNegativeReals)
model.x_B2 = Var(domain=NonNegativeReals)
model.x_B3 = Var(domain=NonNegativeReals)
model.x_B4 = Var(domain=NonNegativeReals)
model.x_E1 = Var(domain=NonNegativeReals)
model.x_E2 = Var(domain=NonNegativeReals)
model.y_B1 = Var(domain=Binary)
model.y_B2 = Var(domain=Binary)
model.y_B3 = Var(domain=Binary)
model.y_B4 = Var(domain=Binary)
model.y_E1 = Var(domain=Binary)
model.y_E2 = Var(domain=Binary)

# Define the objective function
def objective_rule(model):
    return (model.S_A + model.C_A * model.x_A / 1000 +
            model.S_B1 * model.y_B1 + model.C_B1 * model.x_B1 / 1000 +
            model.S_B2 * model.y_B2 + model.C_B2 * model.x_B2 / 1000 +
            model.S_B3 * model.y_B3 + model.C_B3 * model.x_B3 / 1000 +
            model.S_B4 * model.y_B4 + model.C_B4 * model.x_B4 / 1000 +
            model.S_C + model.C_C * model.x_C / 1000 +
            model.S_D + model.C_D * model.x_D / 1000 +
            model.S_E1 * model.y_E1 + model.C_E1 * model.x_E1 / 1000 +
            model.S_E2 * model.y_E2 + (model.C_E2 - model.R_E * (model.x_E2 / 1000000 - 42)) * model.x_E2 / 1000)

model.objective = Objective(rule=objective_rule, sense=minimize)

# Define the constraints
def total_quantity_constraint(model):
    return (model.x_A + model.x_B1 + model.x_B2 + model.x_B3 + model.x_B4 + model.x_C + model.x_D + model.x_E1 + model

model.total_quantity = Constraint(rule=total_quantity_constraint)
```

4. Input Problem Data and Test Model Code

```
%%capture
import sys
import os

if 'google.colab' in sys.modules:
    !pip install idaes-pse --pre
    !idaes get-extensions --to ./bin
    os.environ['PATH'] += ':bin'
```
```

```
from pyomo.environ import *

# Create the model
model = ConcreteModel()

# Define the parameters
model.Q_total = 239600480
model.S_A = 3855.34
model.S_C = 13456.00
model.S_D = 6583.98
model.S_B1, model.S_B2, model.S_B3, model.S_B4 = 125804.84, 269304.84, 464304.84, 761304.84
model.S_E1, model.S_E2 = 0, 0
model.C_A, model.C_C, model.C_D = 61.150, 62.019, 72.488
model.C_B1, model.C_B2, model.C_B3, model.C_B4 = 68.099, 66.049, 64.099, 62.119
model.C_E1, model.C_E2 = 70.150, 68.150
model.R_E = 0.05 / 100  # 0.05%

# Define the decision variables
model.x_A = Var(domain=NonNegativeReals)
model.x_C = Var(domain=NonNegativeReals)
model.x_D = Var(domain=NonNegativeReals)
model.x_B1 = Var(domain=NonNegativeReals)
model.x_B2 = Var(domain=NonNegativeReals)
model.x_B3 = Var(domain=NonNegativeReals)
model.x_B4 = Var(domain=NonNegativeReals)
model.x_E1 = Var(domain=NonNegativeReals)
model.x_E2 = Var(domain=NonNegativeReals)
model.y_B1 = Var(domain=Binary)
model.y_B2 = Var(domain=Binary)
model.y_B3 = Var(domain=Binary)
model.y_B4 = Var(domain=Binary)
model.y_E1 = Var(domain=Binary)
model.y_E2 = Var(domain=Binary)

# Define the objective function
def objective_rule(model):
    return (model.S_A + model.C_A * model.x_A / 1000 +
            model.S_B1 * model.y_B1 + model.C_B1 * model.x_B1 / 1000 +
            model.S_B2 * model.y_B2 + model.C_B2 * model.x_B2 / 1000 +
            model.S_B3 * model.y_B3 + model.C_B3 * model.x_B3 / 1000 +
            model.S_B4 * model.y_B4 + model.C_B4 * model.x_B4 / 1000 +
            model.S_C + model.C_C * model.x_C / 1000 +
            model.S_D + model.C_D * model.x_D / 1000 +
            model.S_E1 * model.y_E1 + model.C_E1 * model.x_E1 / 1000 +
            model.S_E2 * model.y_E2 + (model.C_E2 - model.R_E * (model.x_E2 / 1000000 - 42)) * model.x_E2 / 1000)

model.objective = Objective(rule=objective_rule, sense=minimize)

# Define the constraints
def total_quantity_constraint(model):
    return (model.x_A + model.x_B1 + model.x_B2 + model.x_B3 + model.x_B4 + model.x_C + model.x_D + model.x_E1 + model.x_E2 =

model.total_quantity = Constraint(rule=total_quantity_constraint)

model.vendor_capacity_A = Constraint(expr=model.x_A <= 33000000)
model.vendor_capacity_B1 = Constraint(expr=model.x_B1 <= 70000000 * model.y_B1)
model.vendor_capacity_B2 = Constraint(expr=model.x_B2 <= 100000000 * model.y_B2)
model.vendor_capacity_B3 = Constraint(expr=model.x_B3 <= 150000000 * model.y_B3)
model.vendor_capacity_B4 = Constraint(expr=model.x_B4 <= 160000000 * model.y_B4)
model.vendor_capacity_C = Constraint(expr=model.x_C <= 165600000)
model.vendor_capacity_D = Constraint(expr=model.x_D <= 12000000)
model.vendor_capacity_E1 = Constraint(expr=model.x_E1 <= 42000000 * model.y_E1)
model.vendor_capacity_E2 = Constraint(expr=model.x_E2 <= 77000000 * model.y_E2)

model.vendor_B_range1 = Constraint(expr=22000000 * model.y_B1 <= model.x_B1)
model.vendor_B_range2 = Constraint(expr=70000001 * model.y_B2 <= model.x_B2)
model.vendor_B_range3 = Constraint(expr=100000001 * model.y_B3 <= model.x_B3)
model.vendor_B_range4 = Constraint(expr=150000001 * model.y_B4 <= model.x_B4)
model.vendor_B_selection = Constraint(expr=model.y_B1 + model.y_B2 + model.y_B3 + model.y_B4 <= 1)

model.vendor_E_range1 = Constraint(expr=model.x_E1 <= 42000000 * model.y_E1)
model.vendor_E_range2 = Constraint(expr=42000001 * model.y_E2 <= model.x_E2)
model.vendor_E_selection = Constraint(expr=model.y_E1 + model.y_E2 <= 1)

# Solve the model
solver = SolverFactory('couenne')
result = solver.solve(model)

# Print the results
print("Objective value:", value(model.objective))
print("Quantity from vendor A:", value(model.x_A))
print("Quantity from vendor B (range 1):", value(model.x_B1))
```

```
print("Quantity from vendor B (range 1):", value(model.x_B1))
print("Quantity from vendor B (range 2):", value(model.x_B2))
print("Quantity from vendor B (range 3):", value(model.x_B3))
print("Quantity from vendor B (range 4):", value(model.x_B4))
print("Quantity from vendor C:", value(model.x_C))
print("Quantity from vendor D:", value(model.x_D))
print("Quantity from vendor E (range 1):", value(model.x_E1))
print("Quantity from vendor E (range 2):", value(model.x_E2))
```

```
Objective value: 15197371.072000096
Quantity from vendor A: 32000479.99998664
Quantity from vendor B (range 1): 0.0
Quantity from vendor B (range 2): 0.0
Quantity from vendor B (range 3): 0.0
Quantity from vendor B (range 4): 0.0
Quantity from vendor C: 165600000.0
Quantity from vendor D: 0.0
Quantity from vendor E (range 1): 41999999.99999997
Quantity from vendor E (range 2): 1.3381242752075194e-05
```

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