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"),
)
```

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)
```

```
For each vendor i (A, B, C, D, E):
- Ci: Setup cost for vendor i
- Pi: Unit cost per thousand units for vendor i
- Ui: Maximum number of units that can be supplied by vendor i

For vendor B:
- CB1, CB2, CB3, CB4: Setup costs for different quantity ranges
- PB1, PB2, PB3, PB4: Unit costs per thousand units for different quantity ranges
- LB1, LB2, LB3, LB4: Lower bounds for different quantity ranges
- UB1, UB2, UB3, UB4: Upper bounds for different quantity ranges

For vendor E:
- PE1: Unit cost per thousand units for quantity range 1
- PE2_base: Base unit cost per thousand units for quantity range 2
- PE2_dec: Percentage decrease in unit cost per million units for quantity range 2
- LE1, LE2: Lower bounds for different quantity ranges
- UE1, UE2: Upper bounds for different quantity ranges

Decision Variables:
- xi: Number of units to be purchased from vendor i (continuous)
- yB1, yB2, yB3, yB4: Binary variables indicating the selected quantity range for vendor B
- yE1, yE2: Binary variables indicating the selected quantity range for vendor E

Objective Function:
Minimize:
CA * xA +
CB1 * yB1 + CB2 * yB2 + CB3 * yB3 + CB4 * yB4 +
(PB1 * xB * yB1 + PB2 * xB * yB2 + PB3 * xB * yB3 + PB4 * xB * yB4) / 1000 +
CC * xC +
CD * xD +
(PE1 * xE * yE1 + (PE2_base - PE2_dec * (xE - LE2) / 1000000) * xE * yE2) / 1000

Constraints:
1. Total quantity purchased must equal the required amount:
   xA + xB + xC + xD + xE = T

2. Quantity purchased from each vendor must be within their capacity:
   xA <= UA
   xB <= UB4
   xC <= UC
   xD <= UD
   xE <= UE2

3. Only one quantity range can be selected for vendor B:
   yB1 + yB2 + yB3 + yB4 = 1
   LB1 * yB1 <= xB <= UB1 * yB1
   LB2 * yB2 <= xB <= UB2 * yB2
   LB3 * yB3 <= xB <= UB3 * yB3
   LB4 * yB4 <= xB <= UB4 * yB4

4. Only one quantity range can be selected for vendor E:
   yE1 + yE2 = 1
   LE1 * yE1 <= xE <= UE1 * yE1
   LE2 * yE2 <= xE <= UE2 * yE2

5. Non-negativity constraints:
   xA, xB, xC, xD, xE >= 0
   yB1, yB2, yB3, yB4, yE1, yE2 are binary variables
```

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.xB = Var(domain=NonNegativeReals)
model.xC = Var(domain=NonNegativeReals)
model.xD = Var(domain=NonNegativeReals)
model.xE = Var(domain=NonNegativeReals)
model.yB1 = Var(domain=Binary)
model.yB2 = Var(domain=Binary)
model.yB3 = Var(domain=Binary)
model.yB4 = Var(domain=Binary)
model.yE1 = Var(domain=Binary)
model.yE2 = Var(domain=Binary)

# Define objective function
def obj_rule(model):
    return (
        model.CA * model.xA +
        model.CB1 * model.yB1 + model.CB2 * model.yB2 + model.CB3 * model.yB3 + model.CB4 * model.yB4 +
        (model.PB1 * model.xB * model.yB1 + model.PB2 * model.xB * model.yB2 +
         model.PB3 * model.xB * model.yB3 + model.PB4 * model.xB * model.yB4) / 1000 +
        model.CC * model.xC +
        model.CD * model.xD +
        (model.PE1 * model.xE * model.yE1 +
         (model.PE2_base - model.PE2_dec * (model.xE - model.LE2) / 1000000) * model.xE * model.yE2) / 1000
    )
model.obj = Objective(rule=obj_rule, sense=minimize)

# Define constraints
model.total_quantity = Constraint(expr=model.xA + model.xB + model.xC + model.xD + model.xE == model.T)
model.vendor_capacity_A = Constraint(expr=model.xA <= model.UA)
model.vendor_capacity_B = Constraint(expr=model.xB <= model.UB4)
model.vendor_capacity_C = Constraint(expr=model.xC <= model.UC)
model.vendor_capacity_D = Constraint(expr=model.xD <= model.UD)
model.vendor_capacity_E = Constraint(expr=model.xE <= model.UE2)
model.vendor_B_range = Constraint(expr=model.yB1 + model.yB2 + model.yB3 + model.yB4 == 1)
model.vendor_B_range_1 = Constraint(expr=model.LB1 * model.yB1 <= model.xB <= model.UB1 * model.yB1)
model.vendor_B_range_2 = Constraint(expr=model.LB2 * model.yB2 <= model.xB <= model.UB2 * model.yB2)
model.vendor_B_range_3 = Constraint(expr=model.LB3 * model.yB3 <= model.xB <= model.UB3 * model.yB3)
model.vendor_B_range_4 = Constraint(expr=model.LB4 * model.yB4 <= model.xB <= model.UB4 * model.yB4)
model.vendor_E_range = Constraint(expr=model.yE1 + model.yE2 == 1)
model.vendor_E_range_1 = Constraint(expr=model.LE1 * model.yE1 <= model.xE <= model.UE1 * model.yE1)
model.vendor_E_range_2 = Constraint(expr=model.LE2 * model.yE2 <= model.xE <= model.UE2 * model.yE2)

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

# Print the optimal solution
print("Optimal solution:")
print("xA =", model.xA.value)
print("xB =", model.xB.value)
print("xC =", model.xC.value)
print("xD =", model.xD.value)
print("xE =", model.xE.value)
print("yB1 =", model.yB1.value)
print("yB2 =", model.yB2.value)
print("yB3 =", model.yB3.value)
print("yB4 =", model.yB4.value)
print("yE1 =", model.yE1.value)
```

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 a Pyomo model
model = ConcreteModel()

# Define parameters
model.T = Param(initialize=239600480)  # Sample data: Total number of units required

# Vendor A parameters
model.CA = Param(initialize=3855.34)  # Sample data: Setup cost for vendor A
model.PA = Param(initialize=61.150)  # Sample data: Unit cost per thousand units for vendor A
model.UA = Param(initialize=33000000)  # Sample data: Maximum number of units that can be supplied by vendor A

# Vendor B parameters
model.CB1 = Param(initialize=125804.84)  # Sample data: Setup cost for quantity range 1
model.CB2 = Param(initialize=269304.84)  # Sample data: Setup cost for quantity range 2
model.CB3 = Param(initialize=464304.84)  # Sample data: Setup cost for quantity range 3
model.CB4 = Param(initialize=761304)  # Sample data: Setup cost for quantity range 4
model.PB1 = Param(initialize=68.099)  # Sample data: Unit cost per thousand units for quantity range 1
model.PB2 = Param(initialize=66.049)  # Sample data: Unit cost per thousand units for quantity range 2
model.PB3 = Param(initialize=64.099)  # Sample data: Unit cost per thousand units for quantity range 3
model.PB4 = Param(initialize=62.119)  # Sample data: Unit cost per thousand units for quantity range 4
model.LB1 = Param(initialize=22000000)  # Sample data: Lower bound for quantity range 1
model.LB2 = Param(initialize=70000001)  # Sample data: Lower bound for quantity range 2
model.LB3 = Param(initialize=100000001)  # Sample data: Lower bound for quantity range 3
model.LB4 = Param(initialize=150000001)  # Sample data: Lower bound for quantity range 4
model.UB1 = Param(initialize=70000000)  # Sample data: Upper bound for quantity range 1
model.UB2 = Param(initialize=100000000)  # Sample data: Upper bound for quantity range 2
model.UB3 = Param(initialize=150000000)  # Sample data: Upper bound for quantity range 3
model.UB4 = Param(initialize=1600000000)  # Sample data: Upper bound for quantity range 4

# Vendor C parameters
model.CC = Param(initialize=13456.00)  # Sample data: Setup cost for vendor C
model.PC = Param(initialize=62.019)  # Sample data: Unit cost per thousand units for vendor C
model.UC = Param(initialize=165600000)  # Sample data: Maximum number of units that can be supplied by vendor C

# Vendor D parameters
model.CD = Param(initialize=6583.98)  # Sample data: Setup cost for vendor D
model.PD = Param(initialize=72.488)  # Sample data: Unit cost per thousand units for vendor D
model.UD = Param(initialize=12000000)  # Sample data: Maximum number of units that can be supplied by vendor D

# Vendor E parameters
model.PE1 = Param(initialize=70.150)  # Sample data: Unit cost per thousand units for quantity range 1
model.PE2_base = Param(initialize=68.150)  # Sample data: Base unit cost per thousand units for quantity range 2
model.PE2_dec = Param(initialize=0.034075)  # Sample data: Percentage decrease in unit cost per million units for quantity r
model.LE1 = Param(initialize=0)  # Sample data: Lower bound for quantity range 1
model.LE2 = Param(initialize=42000001)  # Sample data: Lower bound for quantity range 2
model.UE1 = Param(initialize=42000000)  # Sample data: Upper bound for quantity range 1
model.UE2 = Param(initialize=77000000)  # Sample data: Upper bound for quantity range 2

# Define decision variables
model.xA = Var(domain=NonNegativeReals)
model.xB = Var(domain=NonNegativeReals)
model.xC = Var(domain=NonNegativeReals)
model.xD = Var(domain=NonNegativeReals)
model.xE = Var(domain=NonNegativeReals)
model.yB1 = Var(domain=Binary)
model.yB2 = Var(domain=Binary)
model.yB3 = Var(domain=Binary)
model.yB4 = Var(domain=Binary)
model.yE1 = Var(domain=Binary)
model.yE2 = Var(domain=Binary)

# Define objective function
def obj_rule(model):
    return (
        model.CA * model.xA +
        model.CB1 * model.yB1 + model.CB2 * model.yB2 + model.CB3 * model.yB3 + model.CB4 * model.yB4 +
        (model.PB1 * model.xB * model.yB1 + model.PB2 * model.xB * model.yB2 +
         model.PB3 * model.xB * model.yB3 + model.PB4 * model.xB * model.yB4) / 1000 +
        model.CC * model.xC +
        model.CD * model.xD +
        (model.PE1 * model.xE * model.yE1 +
         (model.PE2_base - model.PE2_dec * (model.xE - model.LE2) / 1000000) * model.xE * model.yE2) / 1000
    )
model.obj = Objective(rule=obj_rule, sense=minimize)

# Define constraints
model.total_quantity = Constraint(expr=model.xA + model.xB + model.xC + model.xD + model.xE == model.T)
model.vendor_capacity_A = Constraint(expr=model.xA <= model.UA)
model.vendor_capacity_B = Constraint(expr=model.xB <= model.UB4)
model.vendor_capacity_C = Constraint(expr=model.xC <= model.UC)
```

```
model.vendor_capacity_D = Constraint(expr=model.xD <= model.UD)
model.vendor_capacity_E = Constraint(expr=model.xE <= model.UE2)
model.vendor_B_range = Constraint(expr=model.yB1 + model.yB2 + model.yB3 + model.yB4 == 1)
model.vendor_B_range_1 = Constraint(expr=model.LB1 * model.yB1 <= model.xB <= model.UB1 * model.yB1)
model.vendor_B_range_2 = Constraint(expr=model.LB2 * model.yB2 <= model.xB <= model.UB2 * model.yB2)
model.vendor_B_range_3 = Constraint(expr=model.LB3 * model.yB3 <= model.xB <= model.UB3 * model.yB3)
model.vendor_B_range_4 = Constraint(expr=model.LB4 * model.yB4 <= model.xB <= model.UB4 * model.yB4)
model.vendor_E_range = Constraint(expr=model.yE1 + model.yE2 == 1)
model.vendor_E_range_1 = Constraint(expr=model.LE1 * model.yE1 <= model.xE <= model.UE1 * model.yE1)
model.vendor_E_range_2 = Constraint(expr=model.LE2 * model.yE2 <= model.xE <= model.UE2 * model.yE2)

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

# Print the optimal solution
print("Optimal solution:")
print("xA =", model.xA.value)
print("xB =", model.xB.value)
print("xC =", model.xC.value)
print("xD =", model.xD.value)
print("xE =", model.xE.value)
print("yB1 =", model.yB1.value)
print("yB2 =", model.yB2.value)
print("yB3 =", model.yB3.value)
print("yB4 =", model.yB4.value)
print("yE1 =", model.yE1.value)
print("yE2 =", model.yE2.value)
print("Minimum total cost:", model.obj())
```

```
---------------------------------------------------------------------------
PyomoException                            Traceback (most recent call last)
<ipython-input-12-7a20303882a7> in <cell line: 86>()
     84 model.vendor_capacity_E = Constraint(expr=model.xE <= model.UE2)
     85 model.vendor_B_range = Constraint(expr=model.yB1 + model.yB2 +
model.yB3 + model.yB4 == 1)
---> 86 model.vendor_B_range_1 = Constraint(expr=model.LB1 * model.yB1 <=
model.xB <= model.UB1 * model.yB1)
     87 model.vendor_B_range_2 = Constraint(expr=model.LB2 * model.yB2 <=
model.xB <= model.UB2 * model.yB2)
     88 model.vendor_B_range_3 = Constraint(expr=model.LB3 * model.yB3 <=
model.xB <= model.UB3 * model.yB3)

/usr/local/lib/python3.10/dist-packages/pyomo/core/expr/relational_expr.py in
__bool__(self)
     45         if self.is_constant():
     46             return bool(self())
---> 47         raise PyomoException(
     48             """
     49 Cannot convert non-constant Pyomo expression (%s) to bool.

PyomoException: Cannot convert non-constant Pyomo expression (22000000*yB1
<=  xB) to bool.
This error is usually caused by using a Var, unit, or mutable Param in a
Boolean context such as an "if" statement, or when checking container
membership or equality. For example,
    >>> m.x = Var()
    >>> if m.x >= 1:
    ...     pass
and
```

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

```
from pyomo.environ import *

# Create a Pyomo model
model = ConcreteModel()

# Define parameters
model.T = Param(initialize=239600480)  # Sample data: Total number of units required

# Vendor A parameters
model.CA = Param(initialize=3855.34)  # Sample data: Setup cost for vendor A
model.PA = Param(initialize=61.150)  # Sample data: Unit cost per thousand units for vendor A
model.UA = Param(initialize=33000000)  # Sample data: Maximum number of units that can be supplied by vendor A

# Vendor B parameters
model.CB1 = Param(initialize=125804.84)  # Sample data: Setup cost for quantity range 1
model.CB2 = Param(initialize=269304.84)  # Sample data: Setup cost for quantity range 2
model.CB3 = Param(initialize=464304.84)  # Sample data: Setup cost for quantity range 3
model.CB4 = Param(initialize=761304)  # Sample data: Setup cost for quantity range 4
model.PB1 = Param(initialize=68.099)  # Sample data: Unit cost per thousand units for quantity range 1
model.PB2 = Param(initialize=66.049)  # Sample data: Unit cost per thousand units for quantity range 2
```

```
model.PB2 = Param(initialize=66.049)  # Sample data: Unit cost per thousand units for quantity range 2
model.PB3 = Param(initialize=64.099)  # Sample data: Unit cost per thousand units for quantity range 3
model.PB4 = Param(initialize=62.119)  # Sample data: Unit cost per thousand units for quantity range 4
model.LB1 = Param(initialize=22000000)  # Sample data: Lower bound for quantity range 1
model.LB2 = Param(initialize=70000001)  # Sample data: Lower bound for quantity range 2
model.LB3 = Param(initialize=100000001)  # Sample data: Lower bound for quantity range 3
model.LB4 = Param(initialize=150000001)  # Sample data: Lower bound for quantity range 4
model.UB1 = Param(initialize=70000000)  # Sample data: Upper bound for quantity range 1
model.UB2 = Param(initialize=100000000)  # Sample data: Upper bound for quantity range 2
model.UB3 = Param(initialize=150000000)  # Sample data: Upper bound for quantity range 3
model.UB4 = Param(initialize=1600000000)  # Sample data: Upper bound for quantity range 4

# Vendor C parameters
model.CC = Param(initialize=13456.00)  # Sample data: Setup cost for vendor C
model.PC = Param(initialize=62.019)  # Sample data: Unit cost per thousand units for vendor C
model.UC = Param(initialize=165600000)  # Sample data: Maximum number of units that can be supplied by vendor C

# Vendor D parameters
model.CD = Param(initialize=6583.98)  # Sample data: Setup cost for vendor D
model.PD = Param(initialize=72.488)  # Sample data: Unit cost per thousand units for vendor D
model.UD = Param(initialize=12000000)  # Sample data: Maximum number of units that can be supplied by vendor D

# Vendor E parameters
model.PE1 = Param(initialize=70.150)  # Sample data: Unit cost per thousand units for quantity range 1
model.PE2_base = Param(initialize=68.150)  # Sample data: Base unit cost per thousand units for quantity range 2
model.PE2_dec = Param(initialize=0.034075)  # Sample data: Percentage decrease in unit cost per million units for quantity ra
model.LE1 = Param(initialize=0)  # Sample data: Lower bound for quantity range 1
model.LE2 = Param(initialize=42000001)  # Sample data: Lower bound for quantity range 2
model.UE1 = Param(initialize=42000000)  # Sample data: Upper bound for quantity range 1
model.UE2 = Param(initialize=77000000)  # Sample data: Upper bound for quantity range 2

# Define decision variables
model.xA = Var(domain=NonNegativeReals)
model.xB = Var(domain=NonNegativeReals)
model.xC = Var(domain=NonNegativeReals)
model.xD = Var(domain=NonNegativeReals)
model.xE = Var(domain=NonNegativeReals)
model.yB1 = Var(domain=Binary)
model.yB2 = Var(domain=Binary)
model.yB3 = Var(domain=Binary)
model.yB4 = Var(domain=Binary)
model.yE1 = Var(domain=Binary)
model.yE2 = Var(domain=Binary)

# Define objective function
def obj_rule(model):
    return (
        model.CA * model.xA +
        model.CB1 * model.yB1 + model.CB2 * model.yB2 + model.CB3 * model.yB3 + model.CB4 * model.yB4 +
        (model.PB1 * model.xB * model.yB1 + model.PB2 * model.xB * model.yB2 +
         model.PB3 * model.xB * model.yB3 + model.PB4 * model.xB * model.yB4) / 1000 +
        model.CC * model.xC +
        model.CD * model.xD +
        (model.PE1 * model.xE * model.yE1 +
         (model.PE2_base - model.PE2_dec * (model.xE - model.LE2) / 1000000) * model.xE * model.yE2) / 1000
    )
model.obj = Objective(rule=obj_rule, sense=minimize)

# Define constraints
model.total_quantity = Constraint(expr=model.xA + model.xB + model.xC + model.xD + model.xE == model.T)
model.vendor_capacity_A = Constraint(expr=model.xA <= model.UA)
model.vendor_capacity_B = Constraint(expr=model.xB <= model.UB4)
model.vendor_capacity_C = Constraint(expr=model.xC <= model.UC)
model.vendor_capacity_D = Constraint(expr=model.xD <= model.UD)
model.vendor_capacity_E = Constraint(expr=model.xE <= model.UE2)
model.vendor_B_range = Constraint(expr=model.yB1 + model.yB2 + model.yB3 + model.yB4 == 1)
model.vendor_B_range_1_lower = Constraint(expr=model.LB1 * model.yB1 <= model.xB)
model.vendor_B_range_1_upper = Constraint(expr=model.xB <= model.UB1 * model.yB1)
model.vendor_B_range_2_lower = Constraint(expr=model.LB2 * model.yB2 <= model.xB)
```