## 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
# Variables Prompt
prompt11_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt11_MathematicalModel.txt'

# Objective Prompt
prompt12_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt12_MathematicalModel.txt'

# Constraint Prompt
prompt13_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt13_MathematicalModel.txt'

# Code Prompt
prompt2_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt2_PyomoCode.txt'
problem_desc_path = '/content/drive/MyDrive/Thesis/ProblemDescriptions/NL/NL3.txt'

prompt11_file = open(prompt11_path, "r")
prompt12_file = open(prompt12_path, "r")
prompt13_file = open(prompt13_path, "r")
prompt2_file = open(prompt2_path, "r")
problem_desc_file = open(problem_desc_path, "r")

prompt11 = prompt11_file.read()
print("Prompt 1.1 (Variables):\n", prompt11)

prompt12 = prompt12_file.read()
print("Prompt 1.2 (Objctive):\n", prompt12)

prompt13 = prompt13_file.read()
print("Prompt 1.3 (Constraints):\n", prompt13)

prompt2 = prompt2_file.read()
print("Prompt 2:\n", prompt2)

problem_desc = problem_desc_file.read()
print("Problem Description:\n", problem_desc)
```

```
Prompt 1.1 (Variables):
 Please formulate only the variables for this mathematical optimization problem.
Prompt 1.2 (Objctive):
 Please formulate only the objective function for this mathematical optimization problem.
Prompt 1.3 (Constraints):
 Please formulate only the constraints for this mathematical optimization problem.
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
```

```
Collecting anthropic
  Downloading anthropic-0.28.0-py3-none-any.whl (862 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 862.7/862.7 kB 9.5 MB/s eta 0:00:00
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-packages (from anthropic) (1.7.0)
Collecting httpx<1,>=0.23.0 (from anthropic)
  Downloading httpx-0.27.0-py3-none-any.whl (75 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 9.6 MB/s eta 0:00:00
Collecting jiter<1,>=0.4.0 (from anthropic)
  Downloading jiter-0.4.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (328 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 328.3/328.3 kB 32.5 MB/s eta 0:00:00
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (2.7.3)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from anthropic) (1.3.1)
Requirement already satisfied: tokenizers>=0.13.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (0.19.1)
Requirement already satisfied: typing-extensions<5,>=4.7 in /usr/local/lib/python3.10/dist-packages (from anthropic) (4.
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->anthropic) (3
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->anthropi
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->anthropic) (20
Collecting httpcore==1.* (from httpx<1,>=0.23.0->anthropic)
  Downloading httpcore-1.0.5-py3-none-any.whl (77 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 10.4 MB/s eta 0:00:00
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->anthropic)
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 3.3 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.
Requirement already satisfied: pydantic-core==2.18.4 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.0
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/lib/python3.10/dist-packages (from tokenizers>
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->t
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->t
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->huggi
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface
Installing collected packages: jiter, h11, httpcore, httpx, anthropic
Successfully installed anthropic-0.28.0 h11-0.14.0 httpcore-1.0.5 httpx-0.27.0 jiter-0.4.1
```

```
# 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 1.1: Create Variables for Mathematical Model

```
message11 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt11,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)

response_p11 = message11.content[0].text


# Print response
print(response_p11)
```

```
Okay, let's define the variables for this optimization problem:

Let xA be the number of units (in thousands) purchased from vendor A.
Let xB1, xB2, xB3, xB4 be the number of units (in thousands) purchased from vendor B in the quantity ranges 22,000,000-7
Let xC be the number of units (in thousands) purchased from vendor C.
Let xD be the number of units (in thousands) purchased from vendor D.
```

```
Let xE1 be the number of units (in thousands) purchased from vendor E in the 0 to 42,000,000 range.
Let xE2 be the number of units (in thousands) purchased from vendor E in the 42,000,001 to 77,000,000 range.

Also define the following binary variables:

Let yB1 = 1 if 22,000 ≤ xB1 ≤ 70,000, and 0 otherwise.
Let yB2 = 1 if 70,000 < xB2 ≤ 100,000, and 0 otherwise.
Let yB3 = 1 if 100,000 < xB3 ≤ 150,000, and 0 otherwise.
Let yB4 = 1 if 150,000 < xB4 ≤ 160,000, and 0 otherwise.
Let yE2 = 1 if 42,000 < xE2 ≤ 77,000, and 0 otherwise.

The binary variables are needed to model the fixed costs and quantity discounts that apply only if certain quantity rang
```

## 1. Prompt 1.2: Create Objective for Mathematical Model

```
message12 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt12,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11)}
    ]
)

response_p12 = message12.content[0].text


# Print response
print(response_p12)
```

```
The objective function for this optimization problem can be formulated as follows:

Minimize:
3855.34*yA + 61.150*xA
+ 125804.84*yB1 + 68.099*xB1
+ 269304.84*yB2 + 66.049*xB2
+ 464304.84*yB3 + 64.099*xB3
+ 761304.84*yB4 + 62.119*xB4
+ 13456*yC + 62.019*xC
+ 6583.98*yD + 72.488*xD
+ 70.150*xE1 + 84000*yE2 + (68.15 - 0.05*(xE2-42000)/1000)*xE2

Here's an explanation of each term:

- For vendor A: 3855.34*yA represents the setup cost, and 61.150*xA represents the variable cost based on the number of
- For vendor B: There are four possible quantity ranges, each with its own setup cost and variable cost. The binary vari
- For vendor C: 13456*yC represents the setup cost, and 62.019*xC represents the variable cost.
- For vendor D: 6583.98*yD represents the setup cost, and 72.488*xD represents the variable cost.
- For vendor E: 70.150*xE1 represents the variable cost for the first quantity range (0 to 42,000,000). For the second r

The goal is to minimize this objective function, which represents the total cost of purchasing the required units from t
```

## 1. Prompt 1.3: Create Constraints for Mathematical Model

```
message13 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt13,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11 + response_p12)}
    ]
)

response_p13 = message13.content[0].text


# Print response
print(response_p13)
```

```
Here are the constraints for this optimization problem:

1. Total quantity constraint:
   xA + xB1 + xB2 + xB3 + xB4 + xC + xD + xE1 + xE2 = 239600.480

2. Vendor capacity constraints:
   xA ≤ 33000
   xB1 + xB2 + xB3 + xB4 ≤ 160000
   xC ≤ 165600
   xD ≤ 12000
```

```
   xE1 + xE2 ≤ 77000

3. Vendor B quantity range constraints:
   22000 * yB1 ≤ xB1 ≤ 70000 * yB1
   70000 * yB2 ≤ xB2 ≤ 100000 * yB2
   100000 * yB3 ≤ xB3 ≤ 150000 * yB3
   150000 * yB4 ≤ xB4 ≤ 160000 * yB4
   yB1 + yB2 + yB3 + yB4 ≤ 1

4. Vendor E quantity range constraints:
   xE1 ≤ 42000
   42000 * yE2 ≤ xE2 ≤ 77000 * yE2

5. Non-negativity constraints:
   xA, xB1, xB2, xB3, xB4, xC, xD, xE1, xE2 ≥ 0

6. Binary constraints:
   yA, yB1, yB2, yB3, yB4, yC, yD, yE2 ∈ {0, 1}

Explanation of constraints:

1. The total quantity purchased from all vendors must equal the required 239,600,480 units.

2. Each vendor has a maximum capacity that cannot be exceeded.

3. For vendor B, the quantity purchased must fall within one of the four ranges, and only one range can be selected (ens

4. For vendor E, the quantity purchased in the second range (xE2) must be between 42,000,001 and 77,000,000 if that rang

5. All quantity variables must be non-negative.

6. The binary variables can only take the values 0 or 1.

These constraints, along with the objective function, define the complete optimization problem for minimizing the total
```

## 2. Prompt 2: Write Code based on Mathematical Model

In case more than the component of each step was generated, use "last instance"/ "newest version" of the components.

Example: Prompt 1.1 (Variables): Model generates only variables Prompt 1.2 (Objectvie): Model generates objective and constraints Prompt 1.3 (Constraints): Model generates only constraints

Use Variables from step 1.1, use objective from 1.2 and use constraints from 1.3 (since constraints version 1.3 is more recent than constraints version 1.2).

This set up allows for potential self-correction of the model throughout the three prompts.

Correctness of each component is still generated in the respective step, so self-correction is independent of this.

```
message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": (response_p11+ response_p12 + response_p13)}
    ]
)

response_p2 = message2.content[0].text


# Print response
print(response_p2)
```

```
model.range_B4 = Constraint(expr=150000 * model.yB4 <= model.xB4 <= 160000 * model.yB4)
model.select_B = Constraint(expr=model.yB1 + model.yB2 + model.yB3 + model.yB4 <= 1)
model.range_E1 = Constraint(expr=model.xE1 <= 42000)
model.range_E2 = Constraint(expr=42000 * model.yE2 <= model.xE2 <= 77000 * model.yE2)

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

# Print the results
print(f"Objective value: {model.obj():.2f}")
print(f"xA: {model.xA.value:.2f}")
print(f"xB1: {model.xB1.value:.2f}")
print(f"xB2: {model.xB2.value:.2f}")
print(f"xB3: {model.xB3.value:.2f}")
print(f"xB4: {model.xB4.value:.2f}")
print(f"xC: {model.xC.value:.2f}")
print(f"xD: {model.xD.value:.2f}")
print(f"xE1: {model.xE1.value:.2f}")
print(f"xE2: {model.xE2.value:.2f}")
```

In this code:

1. We create a Pyomo `ConcreteModel` to represent the optimization problem.

2. We define the decision variables using `Var` and specify their domains (`NonNegativeReals` for continuous variables

3. We define the objective function using `Objective` and specify the expression for the total cost. The `sense` param

4. We define the constraints using `Constraint` and specify the expressions for each constraint.

5. We create a solver instance using `SolverFactory` and specify the solver to be used (in this case, 'glpk').

6. We solve the model using the `solve` method of the solver and store the results.

7. Finally, we print the objective value and the values of the decision variables.

## 4. Input Problem Data and Test Model Code

```
# Installing pyomo nd solver
!pip install -q pyomo
!pip install pandas
!apt-get install -y -qq glpk-utils
!pip install glpk
```

Show hidden output

```
from pyomo.environ import *

# Create a Pyomo model
model = ConcreteModel()

# Decision variables
model.xA = Var(domain=NonNegativeReals)
model.xB1 = Var(domain=NonNegativeReals)
model.xB2 = Var(domain=NonNegativeReals)
model.xB3 = Var(domain=NonNegativeReals)
model.xB4 = Var(domain=NonNegativeReals)
model.xC = Var(domain=NonNegativeReals)
model.xD = Var(domain=NonNegativeReals)
model.xE1 = Var(domain=NonNegativeReals)
model.xE2 = Var(domain=NonNegativeReals)
model.yA = Var(domain=Binary)
model.yB1 = Var(domain=Binary)
model.yB2 = Var(domain=Binary)
model.yB3 = Var(domain=Binary)
model.yB4 = Var(domain=Binary)
model.yC = Var(domain=Binary)
model.yD = Var(domain=Binary)
model.yE2 = Var(domain=Binary)

# Objective function
def obj_rule(model):
    return (
        3855.34 * model.yA + 61.150 * model.xA +
        125804.84 * model.yB1 + 68.099 * model.xB1 +
        269304.84 * model.yB2 + 66.049 * model.xB2 +
        464304.84 * model.yB3 + 64.099 * model.xB3 +
        761304.84 * model.yB4 + 62.119 * model.xB4 +
        13456 * model.yC + 62.019 * model.xC +
        6583.98 * model.yD + 72.488 * model.xD +
        70.150 * model.xE1 + 84000 * model.yE2 + (68.15 - 0.05 * (model.xE2 - 42000) / 1000) * model.xE2
    )
model.obj = Objective(rule=obj_rule, sense=minimize)

# Constraints
model.total_quantity = Constraint(expr=model.xA + model.xB1 + model.xB2 + model.xB3 + model.xB4 + model.xC + model.xD + mode
model.capacity_A = Constraint(expr=model.xA <= 33000)
model.capacity_B = Constraint(expr=model.xB1 + model.xB2 + model.xB3 + model.xB4 <= 160000)
model.capacity_C = Constraint(expr=model.xC <= 165600)
model.capacity_D = Constraint(expr=model.xD <= 12000)
model.capacity_E = Constraint(expr=model.xE1 + model.xE2 <= 77000)
model.range_B1 = Constraint(expr=22000 * model.yB1 <= model.xB1 <= 70000 * model.yB1)
model.range_B2 = Constraint(expr=70000 * model.yB2 <= model.xB2 <= 100000 * model.yB2)
model.range_B3 = Constraint(expr=100000 * model.yB3 <= model.xB3 <= 150000 * model.yB3)
model.range_B4 = Constraint(expr=150000 * model.yB4 <= model.xB4 <= 160000 * model.yB4)
model.select_B = Constraint(expr=model.yB1 + model.yB2 + model.yB3 + model.yB4 <= 1)
model.range_E1 = Constraint(expr=model.xE1 <= 42000)
model.range_E2 = Constraint(expr=42000 * model.yE2 <= model.xE2 <= 77000 * model.yE2)

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

# Print the results
print(f"Objective value: {model.obj():.2f}")
print(f"xA: {model.xA.value:.2f}")
print(f"xB1: {model.xB1.value:.2f}")
print(f"xB2: {model.xB2.value:.2f}")
print(f"xB3: {model.xB3.value:.2f}")
print(f"xB4: {model.xB4.value:.2f}")
print(f"xC: {model.xC.value:.2f}")
print(f"xD: {model.xD.value:.2f}")
print(f"xE1: {model.xE1.value:.2f}")
print(f"xE2: {model.xE2.value:.2f}")
```

```
---------------------------------------------------------------------------
PyomoException                            Traceback (most recent call last)
<ipython-input-15-ef4232942ee0> in <cell line: 46>()
     44 model.capacity_D = Constraint(expr=model.xD <= 12000)
     45 model.capacity_E = Constraint(expr=model.xE1 + model.xE2 <= 77000)
---> 46 model.range_B1 = Constraint(expr=22000 * model.yB1 <= model.xB1 <=
70000 * model.yB1)
     47 model.range_B2 = Constraint(expr=70000 * model.yB2 <= model.xB2 <=
100000 * model.yB2)
     48 model.range_B3 = Constraint(expr=100000 * model.yB3 <= model.xB3 <=
150000 * 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 (22000*yB1  <=
xB1) 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
    >>> m.v = Var()
```

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

```
%%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()

# Decision variables
model.xA = Var(domain=NonNegativeReals)
model.xB1 = Var(domain=NonNegativeReals)
model.xB2 = Var(domain=NonNegativeReals)
model.xB3 = Var(domain=NonNegativeReals)
model.xB4 = Var(domain=NonNegativeReals)
model.xC = Var(domain=NonNegativeReals)
model.xD = Var(domain=NonNegativeReals)
model.xE1 = Var(domain=NonNegativeReals)
model.xE2 = Var(domain=NonNegativeReals)
model.yA = Var(domain=Binary)
model.yB1 = Var(domain=Binary)
model.yB2 = Var(domain=Binary)
model.yB3 = Var(domain=Binary)
model.yB4 = Var(domain=Binary)
model.yC = Var(domain=Binary)
model.yD = Var(domain=Binary)
model.yE2 = Var(domain=Binary)

# Objective function
def obj_rule(model):
    return (
        3855.34 * model.yA + 61.150 * model.xA +
        125804.84 * model.yB1 + 68.099 * model.xB1 +
        269304.84 * model.yB2 + 66.049 * model.xB2 +
        464304.84 * model.yB3 + 64.099 * model.xB3 +
        761304.84 * model.yB4 + 62.119 * model.xB4 +
        13456 * model.yC + 62.019 * model.xC +
        6583.98 * model.yD + 72.488 * model.xD +
        70.150 * model.xE1 + 84000 * model.yE2 + (68.15 - 0.05 * (model.xE2 - 42000) / 1000) * model.xE2
    )
model.obj = Objective(rule=obj_rule, sense=minimize)

# Constraints
model.total_quantity = Constraint(expr=model.xA + model.xB1 + model.xB2 + model.xB3 + model.xB4 + model.xC + model.xD + model
```

```
model.capacity_A = Constraint(expr=model.xA <= 33000)
model.capacity_B = Constraint(expr=model.xB1 + model.xB2 + model.xB3 + model.xB4 <= 160000)
model.capacity_C = Constraint(expr=model.xC <= 165600)
model.capacity_D = Constraint(expr=model.xD <= 12000)
model.capacity_E = Constraint(expr=model.xE1 + model.xE2 <= 77000)
model.range_B1_lower = Constraint(expr=22000 * model.yB1 <= model.xB1)
model.range_B1_upper = Constraint(expr=model.xB1 <= 70000 * model.yB1)
model.range_B2_lower = Constraint(expr=70000 * model.yB2 <= model.xB2)
model.range_B2_upper = Constraint(expr=model.xB2 <= 100000 * model.yB2)
model.range_B3_lower = Constraint(expr=100000 * model.yB3 <= model.xB3)
model.range_B3_upper = Constraint(expr=model.xB3 <= 150000 * model.yB3)
model.range_B4_lower = Constraint(expr=150000 * model.yB4 <= model.xB4)
model.range_B4_upper = Constraint(expr=model.xB4 <= 160000 * model.yB4)
model.select_B = Constraint(expr=model.yB1 + model.yB2 + model.yB3 + model.yB4 <= 1)
model.range_E1 = Constraint(expr=model.xE1 <= 42000)
model.range_E2_lower = Constraint(expr=42000 * model.yE2 <= model.xE2)
model.range_E2_upper = Constraint(expr=model.xE2 <= 77000 * model.yE2)
```