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
```

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 1.1: 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)
```

```
Here are the variables for this mathematical 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 yB be a binary variable that equals 1 if any units are purchased from vendor B, and 0 otherwise.

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 range 0 to 42,000,000.

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

Let zE2 be the number of full millions of units above 42,000,000 purchased from vendor E (this is to calculate the price
```

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)
```

```
Here is the objective function for the given mathematical optimization problem:
```

```
Minimize:
3855.34 + 61.150xA
+ 125804.84yB + 68.099xB1
+ 269304.84yB + 66.049xB2
+ 464304.84yB + 64.099xB3
+ 761304.84yB + 62.119xB4
+ 13456 + 62.019xC
+ 6583.98 + 72.488xD
+ 70.150xE1
+ 84000 + (68.150 - 0.0005*68.150*zE2)xE2

The objective is to minimize the total cost, which includes the setup costs and the variable unit costs for each vendor,
```

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 mathematical optimization problem:

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

2. Vendor capacity constraints:
   xA <= 33000
   xC <= 165600
   xD <= 12000
   xE1 + xE2 <= 77000

3. Vendor B quantity range constraints:
   22000 yB <= xB1 <= 70000 yB
   70000 yB <= xB2 <= 100000 yB
   100000 yB <= xB3 <= 150000 yB
   150000 yB <= xB4 <= 160000 yB

4. Vendor E quantity range constraints:
   0 <= xE1 <= 42000
   42000 <= xE2 <= 77000

5. Vendor B binary variable constraint:
   yB is binary

6. Vendor E price decrease calculation constraint:
   zE2 = floor((xE2 - 42000) / 1000)

7. Non-negativity constraints:
   xA, xB1, xB2, xB3, xB4, xC, xD, xE1, xE2, zE2 >= 0

The constraints ensure that:
1. The total quantity purchased from all vendors equals the required amount.
2. The quantity purchased from each vendor does not exceed their capacity.
3. The quantity ranges for vendor B are respected, and the binary variable yB is used to enforce the setup cost.
4. The quantity ranges for vendor E are respected.
5. The binary variable yB is defined.
6. The variable zE2 is calculated to determine the price decrease for units purchased from vendor E above 42 million.
7. All decision variables are non-negative.
```

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.xD = Var(domain=NonNegativeReals)
model.xE1 = Var(domain=NonNegativeReals)
model.xE2 = Var(domain=NonNegativeReals)
model.zE2 = Var(domain=NonNegativeReals)

# Define objective function
def objective_rule(model):
    return (
        3855.34 + 61.150 * model.xA
        + 125804.84 * model.yB + 68.099 * model.xB1
        + 269304.84 * model.yB + 66.049 * model.xB2
        + 464304.84 * model.yB + 64.099 * model.xB3
        + 761304.84 * model.yB + 62.119 * model.xB4
        + 13456 + 62.019 * model.xC
        + 6583.98 + 72.488 * model.xD
        + 70.150 * model.xE1
        + 84000 + (68.150 - 0.0005 * 68.150 * model.zE2) * model.xE2
    )
model.objective = Objective(rule=objective_rule, sense=minimize)

# Define constraints
model.total_quantity = Constraint(expr=model.xA + model.xB1 + model.xB2 + model.xB3 + model.xB4 + model.xC + model.xD
model.capacity_A = Constraint(expr=model.xA <= 33000)
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.yB <= model.xB1 <= 70000 * model.yB)
model.range_B2 = Constraint(expr=70000 * model.yB <= model.xB2 <= 100000 * model.yB)
model.range_B3 = Constraint(expr=100000 * model.yB <= model.xB3 <= 150000 * model.yB)
model.range_B4 = Constraint(expr=150000 * model.yB <= model.xB4 <= 160000 * model.yB)
model.range_E1 = Constraint(expr=0 <= model.xE1 <= 42000)
model.range_E2 = Constraint(expr=42000 <= model.xE2 <= 77000)
model.price_decrease = Constraint(expr=model.zE2 == floor((model.xE2 - 42000) / 1000))

# Solve the model
solver = SolverFactory('glpk')  # Specify the solver (e.g., 'glpk', 'cplex', 'gurobi')
results = solver.solve(model)

# Print the results
print("Objective value:", value(model.objective))
print("xA =", value(model.xA))
print("xB1 =", value(model.xB1))
print("xB2 =", value(model.xB2))
print("xB3 =", value(model.xB3))
print("xB4 =", value(model.xB4))
print("yB =", value(model.yB))
print("xC =", value(model.xC))
print("xD =", value(model.xD))
print("xE1 =", value(model.xE1))
print("xE2 =", value(model.xE2))
print("zE2 =", value(model.zE2))
```

Please note that this code assumes you have Pyomo installed and a suitable solver (e.g., GLPK, CPLEX, Gurobi) availabl

This code defines the decision variables, objective function, and constraints as per the provided problem description.

The code solves the optimization problem and prints 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()

# Define 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.yB = Var(domain=Binary)
model.xC = Var(domain=NonNegativeReals)
model.xD = Var(domain=NonNegativeReals)
model.xE1 = Var(domain=NonNegativeReals)
model.xE2 = Var(domain=NonNegativeReals)
model.zE2 = Var(domain=NonNegativeReals)

# Define objective function
def objective_rule(model):
    return (
        3855.34 + 61.150 * model.xA
        + 125804.84 * model.yB + 68.099 * model.xB1
        + 269304.84 * model.yB + 66.049 * model.xB2
        + 464304.84 * model.yB + 64.099 * model.xB3
        + 761304.84 * model.yB + 62.119 * model.xB4
        + 13456 + 62.019 * model.xC
        + 6583.98 + 72.488 * model.xD
        + 70.150 * model.xE1
        + 84000 + (68.150 - 0.0005 * 68.150 * model.zE2) * model.xE2
    )
model.objective = Objective(rule=objective_rule, sense=minimize)

# Define 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_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.yB <= model.xB1 <= 70000 * model.yB)
model.range_B2 = Constraint(expr=70000 * model.yB <= model.xB2 <= 100000 * model.yB)
model.range_B3 = Constraint(expr=100000 * model.yB <= model.xB3 <= 150000 * model.yB)
model.range_B4 = Constraint(expr=150000 * model.yB <= model.xB4 <= 160000 * model.yB)
model.range_E1 = Constraint(expr=0 <= model.xE1 <= 42000)
model.range_E2 = Constraint(expr=42000 <= model.xE2 <= 77000)
model.price_decrease = Constraint(expr=model.zE2 == floor((model.xE2 - 42000) / 1000))

# Solve the model
solver = SolverFactory('glpk')  # Specify the solver (e.g., 'glpk', 'cplex', 'gurobi')
results = solver.solve(model)

# Print the results
print("Objective value:", value(model.objective))
print("xA =", value(model.xA))
print("xB1 =", value(model.xB1))
print("xB2 =", value(model.xB2))
print("xB3 =", value(model.xB3))
print("xB4 =", value(model.xB4))
print("yB =", value(model.yB))
print("xC =", value(model.xC))
print("xD =", value(model.xD))
print("xE1 =", value(model.xE1))
print("xE2 =", value(model.xE2))
print("zE2 =", value(model.zE2))
```

```
---------------------------------------------------------------------------
PyomoException                            Traceback (most recent call last)
<ipython-input-16-7e205cbc7de7> in <cell line: 40>()
     38 model.capacity_D = Constraint(expr=model.xD <= 12000)
     39 model.capacity_E = Constraint(expr=model.xE1 + model.xE2 <= 77000)
---> 40 model.range_B1 = Constraint(expr=22000 * model.yB <= model.xB1 <=
70000 * model.yB)
     41 model.range_B2 = Constraint(expr=70000 * model.yB <= model.xB2 <=
100000 * model.yB)
     42 model.range_B3 = Constraint(expr=100000 * model.yB <= model.xB3 <=
150000 * model.yB)

/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*yB  <=
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()

# Define 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.yB = Var(domain=Binary)
model.xC = Var(domain=NonNegativeReals)
model.xD = Var(domain=NonNegativeReals)
model.xE1 = Var(domain=NonNegativeReals)
model.xE2 = Var(domain=NonNegativeReals)
model.zE2_int = Var(domain=NonNegativeIntegers)

# Define objective function
def objective_rule(model):
    return (
        3855.34 + 61.150 * model.xA
        + 125804.84 * model.yB + 68.099 * model.xB1
        + 269304.84 * model.yB + 66.049 * model.xB2
        + 464304.84 * model.yB + 64.099 * model.xB3
        + 761304.84 * model.yB + 62.119 * model.xB4
        + 13456 + 62.019 * model.xC
        + 6583.98 + 72.488 * model.xD
        + 70.150 * model.xE1
        + 84000 + (68.150 - 0.0005 * 68.150 * model.zE2_int) * model.xE2
    )
model.objective = Objective(rule=objective_rule, sense=minimize)

# Define 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_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.yB <= model.xB1)
```

```
model.range_B1_upper = Constraint(expr=model.xB1 <= 70000 * model.yB)
model.range_B2 = Constraint(expr=70000 * model.yB <= model.xB2)
model.range_B2_upper = Constraint(expr=model.xB2 <= 100000 * model.yB)
model.range_B3 = Constraint(expr=100000 * model.yB <= model.xB3)
model.range_B3_upper = Constraint(expr=model.xB3 <= 150000 * model.yB)
model.range_B4 = Constraint(expr=150000 * model.yB <= model.xB4)
model.range_B4_upper = Constraint(expr=model.xB4 <= 160000 * model.yB)
model.range_E1 = Constraint(expr=model.xE1 <= 42000)
model.range_E2 = Constraint(expr=42000 <= model.xE2)
model.range_E2_upper = Constraint(expr=model.xE2 <= 77000)

# Constraints to simulate floor function
model.floor_lower = Constraint(expr=model.zE2_int <= (model.xE2 - 42000) / 1000)
model.floor_upper = Constraint(expr=model.zE2_int + 0.9999999999999999 >= (model.xE2 - 42000) / 1000)


# Solve the model
solver = SolverFactory('couenne')  # Specify the solver (e.g., 'glpk', 'cplex', 'gurobi')
results = solver.solve(model)

# Print the results
print("Objective value:", value(model.objective))
print("xA =", round(value(model.xA), 2))
print("xB1 =", round(value(model.xB1), 2))
print("xB2 =", round(value(model.xB2), 2))
print("xB3 =", round(value(model.xB3), 2))
print("xB4 =", round(value(model.xB4), 2))
print("yB =", round(value(model.yB), 2))
print("xC =", round(value(model.xC), 2))
print("xD =", round(value(model.xD), 2))
print("xE1 =", round(value(model.xE1), 2))
print("xE2 =", round(value(model.xE2), 2))
print("zE2 =", round(value(model.zE2_int), 2))
```

```
Objective value: 15196502.488366047
xA = 33000.0
xB1 = 0.0
xB2 = 0.0
xB3 = 0.0
xB4 = 0.0
yB = 0.0
xC = 164600.48
xD = 0.0
xE1 = -0.0
xE2 = 42000.0
zE2 = 0.0
```