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/NL4.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:
 We are looking at an alkylation process which will include the following 10 variables: olefin feed (barrels per day), i

We want to maximize the daily profit of this alkylation process.
The profit is defined as the revenue generated from the alkylate yield multiplied with the motor octane number, minus th

Relationships in terms of other variables for alkylate yield, motor octane number, acid dilution factor, and F-4 perform
This regression estimate can deviate in both directions from true value of these variables by 2, 1, 5 and 10 percent, re
Alkylate yield is a function of olefin feed and external isobutane-to-olefine yield. Alkalyte yield equals the amount of
The motor octane number is derived from the external isobutane-to-olefin ratio and the acid strength. The motor octane n
The acid dilution factor is calculated based on the F-4 performance number. The acid dillution factor is expressed as 35
Lastly, the F-4 performance number depends on the motor octane number. F-4 performance number is calculated as -133 plus

There are some additional constraints imposed by the nature of the chemical process.
```

```
Each variable has a lower and an upper bound.
The external isobutane-to-olefin ratio needs to equal the ratio of isobutane recycle plus isobutane makeup to olefin fee
The acid strength needs to equal the ratio of 93000 times acid addition rate to acid addition rate multiplied by acid di
Lastly, 1.22 alkylate yield needs to be equal to the combined olefin feed and isobutane makeup.
```

```
!pip install anthropic
```

```
Collecting anthropic
  Downloading anthropic-0.28.0-py3-none-any.whl (862 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 862.7/862.7 kB 10.1 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 8.8 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 13.5 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 8.7 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 6.8 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)
```

```
Great! Let's define the variables for this alkylation process optimization problem. We will use the following notation:

    Variables:
    - O: olefin feed (barrels per day)
    - R: isobutane recycle (barrels per day)
    - A: acid addition rate (thousands of pounds per day)
    - Y: alkylate yield (barrels per day)
    - M: isobutane makeup (barrels per day)
    - S: acid strength (weight per cent)
    - N: motor octane number
    - E: external isobutane-to-olefin ratio
```

```
- D: acid dilution factor
- F: F-4 performance number

Objective Function:
- P: daily profit of the alkylation process

Regression Formulas:
- Ŷ: estimated alkylate yield
- N̂: estimated motor octane number
- D̂: estimated acid dilution factor
- F̂: estimated F-4 performance number

These variables, along with the objective function and regression formulas, will be used to formulate the mathematical o
```

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)
```

```
Based on the given information, the objective function for the alkylation process optimization problem can be formulated

    Maximize:
    P = Ŷ * N̂ - (c1 * O + c2 * R + c3 * A + c4 * M)

    Where:
    - P is the daily profit of the alkylation process
    - Ŷ is the estimated alkylate yield
    - N̂ is the estimated motor octane number
    - c1, c2, c3, and c4 are the respective costs per unit for olefin feed, isobutane recycle, acid addition rate, and isobu

    The estimated values Ŷ and N̂ are calculated using the regression formulas:

    Ŷ = O * (1.12 + 0.13167 * E - 0.00667 * E^2)
    N̂ = 86.35 + 1.098 * E - 0.038 * E^2 + 0.325 * (S - 89)

    The objective function aims to maximize the daily profit by considering the revenue generated from the product of estima

    Note that the objective function is expressed in terms of the estimated values Ŷ and N̂, which are derived from the regre
```

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 the alkylation process optimization problem:

    1. Regression formula constraints:
       - 0.98 * Y ≤ Ŷ ≤ 1.02 * Y
       - 0.99 * N ≤ N̂ ≤ 1.01 * N
       - 0.95 * D ≤ D̂ ≤ 1.05 * D
       - 0.90 * F ≤ F̂ ≤ 1.10 * F

    2. Variable bounds:
       - Omin ≤ O ≤ Omax
       - Rmin ≤ R ≤ Rmax
```

```
   - Amin ≤ A ≤ Amax
   - Ymin ≤ Y ≤ Ymax
   - Mmin ≤ M ≤ Mmax
   - Smin ≤ S ≤ Smax
   - Nmin ≤ N ≤ Nmax
   - Emin ≤ E ≤ Emax
   - Dmin ≤ D ≤ Dmax
   - Fmin ≤ F ≤ Fmax

3. External isobutane-to-olefin ratio constraint:
   - E = (R + M) / O

4. Acid strength constraint:
   - S = (93000 * A) / (A * D̂ + 1000 * A)

5. Material balance constraint:
   - 1.22 * Y = O + M

6. Regression formulas:
   - D̂ = 35.82 - 0.222 * F̂
   - F̂ = -133 + 3 * N̂

Note: The subscripts "min" and "max" represent the lower and upper bounds for each variable, respectively.

These constraints ensure that the regression estimates are within the specified deviation ranges, the variables are with
```

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.D.setub(100)   # Dmax
model.F.setlb(50)    # Fmin
model.F.setub(100)   # Fmax

# Define cost coefficients (using sample data)
c1 = 10  # Cost per unit of olefin feed
c2 = 5   # Cost per unit of isobutane recycle
c3 = 100 # Cost per unit of acid addition rate
c4 = 8   # Cost per unit of isobutane makeup

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

# Print the results
print("Optimal Profit: ", value(model.profit))
print("Optimal Olefin Feed: ", value(model.O))
print("Optimal Isobutane Recycle: ", value(model.R))
print("Optimal Acid Addition Rate: ", value(model.A))
print("Optimal Alkylate Yield: ", value(model.Y))
print("Optimal Isobutane Makeup: ", value(model.M))
print("Optimal Acid Strength: ", value(model.S))
print("Optimal Motor Octane Number: ", value(model.N))
print("Optimal External Isobutane-to-Olefin Ratio: ", value(model.E))
print("Optimal Acid Dilution Factor: ", value(model.D))
print("Optimal F-4 Performance Number: ", value(model.F))
```

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'

# Installing pyomo nd solver
!pip install -q pyomo
!pip install pandas
```

```
from pyomo.environ import *

# Create a concrete model
model = ConcreteModel()

# Define cost coefficients (using sample data)
c1 = 5.04  # Cost per unit of olefin feed
c2 = 0.035   # Cost per unit of isobutane recycle
c3 = 10.0 # Cost per unit of acid addition rate
c4 = 3.36   # Cost per unit of isobutane makeup

# Define the variables
model.O = Var(within=PositiveReals)  # Olefin feed (barrels per day)
model.R = Var(within=PositiveReals)  # Isobutane recycle (barrels per day)
model.A = Var(within=PositiveReals)  # Acid addition rate (thousands of pounds per day)
model.Y = Var(within=PositiveReals)  # Alkylate yield (barrels per day)
model.M = Var(within=PositiveReals)  # Isobutane makeup (barrels per day)
model.S = Var(within=PositiveReals)  # Acid strength (weight per cent)
model.N = Var(within=PositiveReals)  # Motor octane number
model.E = Var(within=PositiveReals)  # External isobutane-to-olefin ratio
model.D = Var(within=PositiveReals)  # Acid dilution factor
model.F = Var(within=PositiveReals)  # F-4 performance number

# Define the objective function
def profit_rule(model):
    Y_hat = model.O * (1.12 + 0.13167 * model.E - 0.00667 * model.E**2)
    N_hat = 86.35 + 1.098 * model.E - 0.038 * model.E**2 + 0.325 * (model.S - 89)
    return Y_hat * N_hat - (c1 * model.O + c2 * model.R + c3 * model.A + c4 * model.M)

model.profit = Objective(rule=profit_rule, sense=maximize)

# Define the constraints
def yield_constraint_rule(model):
    Y_hat = model.O * (1.12 + 0.13167 * model.E - 0.00667 * model.E**2)
    return (0.98 * model.Y, Y_hat, 1.02 * model.Y)

model.yield_constraint = Constraint(rule=yield_constraint_rule)

def octane_constraint_rule(model):
    N_hat = 86.35 + 1.098 * model.E - 0.038 * model.E**2 + 0.325 * (model.S - 89)
    return (0.99 * model.N, N_hat, 1.01 * model.N)

model.octane_constraint = Constraint(rule=octane_constraint_rule)

def dilution_constraint_rule(model):
    D_hat = 35.82 - 0.222 * model.F
    return (0.95 * model.D, D_hat, 1.05 * model.D)

model.dilution_constraint = Constraint(rule=dilution_constraint_rule)

def performance_constraint_rule(model):
    F_hat = -133 + 3 * model.N
    return (0.90 * model.F, F_hat, 1.10 * model.F)

model.performance_constraint = Constraint(rule=performance_constraint_rule)

def external_ratio_constraint_rule(model):
    return model.E == (model.R + model.M) / model.O

model.external_ratio_constraint = Constraint(rule=external_ratio_constraint_rule)

def acid_strength_constraint_rule(model):
    D_hat = 35.82 - 0.222 * model.F
    return model.S == (93000 * model.A) / (model.A * D_hat + 1000 * model.A)

model.acid_strength_constraint = Constraint(rule=acid_strength_constraint_rule)

def material_balance_constraint_rule(model):
    return 1.22 * model.Y == model.O + model.M

model.material_balance_constraint = Constraint(rule=material_balance_constraint_rule)

# Define variable bounds (using sample data)
model.O.setlb(0)  # Omin
model.O.setub(2000)  # Omax
model.R.setlb(0)  # Rmin
model.R.setub(16000)  # Rmax
model.A.setlb(0)     # Amin
model.A.setub(120)    # Amax
model.Y.setlb(0)  # Ymin
model.Y.setub(5000)  # Ymax
model.M.setlb(0)   # Mmin
```

```
model.M.setub(2000)  # Mmax
model.S.setlb(85)    # Smin
model.S.setub(93)    # Smax
model.N.setlb(90)    # Nmin
model.N.setub(95)    # Nmax
model.E.setlb(3)     # Emin
model.E.setub(12)     # Emax
model.D.setlb(1.2)     # Dmin
model.D.setub(4)   # Dmax
model.F.setlb(145)    # Fmin
model.F.setub(162)   # Fmax

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

# Print the results
print("Optimal Profit: ", value(model.profit))
print("Optimal Olefin Feed: ", value(model.O))
print("Optimal Isobutane Recycle: ", value(model.R))
print("Optimal Acid Addition Rate: ", value(model.A))
print("Optimal Alkylate Yield: ", value(model.Y))
print("Optimal Isobutane Makeup: ", value(model.M))
print("Optimal Acid Strength: ", value(model.S))
print("Optimal Motor Octane Number: ", value(model.N))
print("Optimal External Isobutane-to-Olefin Ratio: ", value(model.E))
print("Optimal Acid Dilution Factor: ", value(model.D))
print("Optimal F-4 Performance Number: ", value(model.F))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-377cbf69c4ae> in <cell line: 97>()
     95 # Solve the optimization problem
     96 solver = SolverFactory('ipopt')
---> 97 results = solver.solve(model)
     98
     99 # Print the results

                                 11 frames
/usr/local/lib/python3.10/dist-packages/pyomo/core/base/constraint.py in
_get_range_bound(self, range_arg)
    205         bound = self._expr.arg(range_arg)
    206         if not is_fixed(bound):
--> 207             raise ValueError(
    208                 "Constraint '%s' is a Ranged Inequality with a "
    209                 "variable %s bound.  Cannot normalize the "

ValueError: Constraint 'yield_constraint' is a Ranged Inequality with a
variable lower bound.  Cannot normalize the constraint or send it to a
```

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

```
from pyomo.environ import *

# Create a concrete model
model = ConcreteModel()

# Define cost coefficients (using sample data)
c1 = 5.04  # Cost per unit of olefin feed
c2 = 0.035   # Cost per unit of isobutane recycle
c3 = 10.0 # Cost per unit of acid addition rate
c4 = 3.36   # Cost per unit of isobutane makeup

# Define the variables
model.O = Var(within=PositiveReals)  # Olefin feed (barrels per day)
model.R = Var(within=PositiveReals)  # Isobutane recycle (barrels per day)
model.A = Var(within=PositiveReals)  # Acid addition rate (thousands of pounds per day)
model.Y = Var(within=PositiveReals)  # Alkylate yield (barrels per day)
model.M = Var(within=PositiveReals)  # Isobutane makeup (barrels per day)
model.S = Var(within=PositiveReals)  # Acid strength (weight per cent)
model.N = Var(within=PositiveReals)  # Motor octane number
model.E = Var(within=PositiveReals)  # External isobutane-to-olefin ratio
model.D = Var(within=PositiveReals)  # Acid dilution factor
model.F = Var(within=PositiveReals)  # F-4 performance number

# Define the objective function
def profit_rule(model):
    Y_hat = model.O * (1.12 + 0.13167 * model.E - 0.00667 * model.E**2)
    N_hat = 86.35 + 1.098 * model.E - 0.038 * model.E**2 + 0.325 * (model.S - 89)
    return Y_hat * N_hat - (c1 * model.O + c2 * model.R + c3 * model.A + c4 * model.M)
```

```
model.profit = Objective(rule=profit_rule, sense=maximize)

# Define the constraints
def yield_constraint_rule_lower(model):
    Y_hat = model.O * (1.12 + 0.13167 * model.E - 0.00667 * model.E**2)
    return (0.98 * model.Y <= Y_hat)

model.yield_constraint_lower = Constraint(rule=yield_constraint_rule_lower)


def yield_constraint_rule_upper(model):
    Y_hat = model.O * (1.12 + 0.13167 * model.E - 0.00667 * model.E**2)
    return (Y_hat <= 1.02 * model.Y)

model.yield_constraint_upper = Constraint(rule=yield_constraint_rule_upper)

def octane_constraint_rule_lower(model):
    N_hat = 86.35 + 1.098 * model.E - 0.038 * model.E**2 + 0.325 * (model.S - 89)
    return (0.99 * model.N <= N_hat)

model.octane_constraint_lower = Constraint(rule=octane_constraint_rule_lower)

def octane_constraint_rule_upper(model):
    N_hat = 86.35 + 1.098 * model.E - 0.038 * model.E**2 + 0.325 * (model.S - 89)
    return ( N_hat <= 1.01 * model.N)

model.octane_constraint_upper = Constraint(rule=octane_constraint_rule_upper)

def dilution_constraint_rule_lower(model):
    D_hat = 35.82 - 0.222 * model.F
    return (0.95 * model.D <= D_hat)

model.dilution_constraint_lower = Constraint(rule=dilution_constraint_rule_lower)

def dilution_constraint_rule_upper(model):
    D_hat = 35.82 - 0.222 * model.F
    return (0.95 * model.D <= D_hat)

model.dilution_constraint_upper = Constraint(rule=dilution_constraint_rule_upper)

def performance_constraint_rule(model):
    F_hat = -133 + 3 * model.N
    return (F_hat <= 1.10 * model.F)

model.performance_constraint = Constraint(rule=performance_constraint_rule)

def external_ratio_constraint_rule(model):
    return model.E == (model.R + model.M) / model.O

model.external_ratio_constraint = Constraint(rule=external_ratio_constraint_rule)

def acid_strength_constraint_rule(model):
    D_hat = 35.82 - 0.222 * model.F
    return model.S == (93000 * model.A) / (model.A * D_hat + 1000 * model.A)

model.acid_strength_constraint = Constraint(rule=acid_strength_constraint_rule)

def material_balance_constraint_rule(model):
    return 1.22 * model.Y == model.O + model.M

model.material_balance_constraint = Constraint(rule=material_balance_constraint_rule)

# Define variable bounds (using sample data)
model.O.setlb(0)  # Omin
model.O.setub(2000)  # Omax
model.R.setlb(0)  # Rmin
model.R.setub(16000)  # Rmax
model.A.setlb(0)     # Amin
model.A.setub(120)    # Amax
model.Y.setlb(0)  # Ymin
model.Y.setub(5000)  # Ymax
model.M.setlb(0)   # Mmin
model.M.setub(2000)  # Mmax
model.S.setlb(85)    # Smin
model.S.setub(93)    # Smax
model.N.setlb(90)    # Nmin
model.N.setub(95)    # Nmax
model.E.setlb(3)     # Emin
model.E.setub(12)     # Emax
model.D.setlb(1.2)     # Dmin
model.D.setub(4)   # Dmax
model.F.setlb(145)    # Fmin
model.F.setub(162)   # Fmax
```

```
# Solve the optimization problem
solver = SolverFactory('ipopt')
results = solver.solve(model)

# Print the results
print("Optimal Profit: ", value(model.profit))
print("Optimal Olefin Feed: ", value(model.O))
print("Optimal Isobutane Recycle: ", value(model.R))
print("Optimal Acid Addition Rate: ", value(model.A))
print("Optimal Alkylate Yield: ", value(model.Y))
```