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/NL4.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:
 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
```

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)
```

```
- R: Revenue per barrel of alkylate yield per unit of motor octane number
- C_OF: Cost per barrel of olefin feed
- C_IR: Cost per barrel of isobutane recycle
- C_AA: Cost per thousand pounds of acid addition rate
- C_IM: Cost per barrel of isobutane makeup

Decision Variables:
- OF: Olefin feed (barrels per day)
- IR: Isobutane recycle (barrels per day)
- AA: Acid addition rate (thousands of pounds per day)
- AY: Alkylate yield (barrels per day)
- IM: Isobutane makeup (barrels per day)
- AS: Acid strength (weight percent)
- MON: Motor octane number
- EIO: External isobutane-to-olefin ratio
- ADF: Acid dilution factor
- F4: F-4 performance number

Objective Function:
Maximize Profit = R * AY * MON - (C_OF * OF + C_IR * IR + C_AA * AA + C_IM * IM)

Constraints:
1. Alkylate Yield:
   AY = OF * (1.12 + 0.13167 * EIO - 0.00667 * EIO^2)
   0.98 * AY <= OF * (1.12 + 0.13167 * EIO - 0.00667 * EIO^2) <= 1.02 * AY

2. Motor Octane Number:
   MON = 86.35 + 1.098 * EIO - 0.038 * EIO^2 + 0.325 * (AS - 89)
   0.99 * MON <= 86.35 + 1.098 * EIO - 0.038 * EIO^2 + 0.325 * (AS - 89) <= 1.01 * MON

3. Acid Dilution Factor:
   ADF = 35.82 - 0.222 * F4
   0.95 * ADF <= 35.82 - 0.222 * F4 <= 1.05 * ADF

4. F-4 Performance Number:
   F4 = -133 + 3 * MON
   0.90 * F4 <= -133 + 3 * MON <= 1.10 * F4

5. External Isobutane-to-Olefin Ratio:
   EIO = (IR + IM) / OF

6. Acid Strength:
   AS = (93000 * AA) / (AA * ADF + 1000 * AA)

7. Material Balance:
   1.22 * AY = OF + IM

8. Variable Bounds:
   OF_min <= OF <= OF_max
   IR_min <= IR <= IR_max
   AA_min <= AA <= AA_max
   IM_min <= IM <= IM_max
   AS_min <= AS <= AS_max
   MON_min <= MON <= MON_max
   EIO_min <= EIO <= EIO_max
   ADF_min <= ADF <= ADF_max
   F4_min <= F4 <= F4_max
```

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.f4_performance_number_bounds = Constraint(rule=f4_performance_number_bounds)

def external_isobutane_to_olefin_ratio(model):
    return model.EIO == (model.IR + model.IM) / model.OF
model.external_isobutane_to_olefin_ratio_eq = Constraint(rule=external_isobutane_to_olefin_ratio)

def acid_strength(model):
    return model.AS == (93000 * model.AA) / (model.AA * model.ADF + 1000 * model.AA)
model.acid_strength_eq = Constraint(rule=acid_strength)

def material_balance(model):
    return 1.22 * model.AY == model.OF + model.IM
model.material_balance_eq = Constraint(rule=material_balance)

# Define variable bounds (using sample data)
model.OF_min = Param(initialize=0)
model.OF_max = Param(initialize=1000)
model.IR_min = Param(initialize=0)
model.IR_max = Param(initialize=500)
model.AA_min = Param(initialize=0)
model.AA_max = Param(initialize=50)
model.IM_min = Param(initialize=0)
model.IM_max = Param(initialize=200)
model.AS_min = Param(initialize=85)
model.AS_max = Param(initialize=95)
model.MON_min = Param(initialize=88)
model.MON_max = Param(initialize=95)
model.EIO_min = Param(initialize=5)
model.EIO_max = Param(initialize=15)
model.ADF_min = Param(initialize=10)
model.ADF_max = Param(initialize=30)
model.F4_min = Param(initialize=120)
model.F4_max = Param(initialize=180)

model.OF_bounds = Constraint(rule=lambda model: (model.OF_min, model.OF, model.OF_max))
model.IR_bounds = Constraint(rule=lambda model: (model.IR_min, model.IR, model.IR_max))
model.AA_bounds = Constraint(rule=lambda model: (model.AA_min, model.AA, model.AA_max))
model.IM_bounds = Constraint(rule=lambda model: (model.IM_min, model.IM, model.IM_max))
model.AS_bounds = Constraint(rule=lambda model: (model.AS_min, model.AS, model.AS_max))
model.MON_bounds = Constraint(rule=lambda model: (model.MON_min, model.MON, model.MON_max))
model.EIO_bounds = Constraint(rule=lambda model: (model.EIO_min, model.EIO, model.EIO_max))
model.ADF_bounds = Constraint(rule=lambda model: (model.ADF_min, model.ADF, model.ADF_max))
model.F4_bounds = Constraint(rule=lambda model: (model.F4_min, model.F4, model.F4_max))

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

# Print the results
print(f"Objective Value (Profit): {model.profit()}")
print(f"Olefin Feed (OF): {model.OF()}")
print(f"Isobutane Recycle (IR): {model.IR()}")
print(f"Acid Addition Rate (AA): {model.AA()}")
print(f"Alkylate Yield (AY): {model.AY()}")
print(f"Isobutane Makeup (IM): {model.IM()}")
print(f"Acid Strength (AS): {model.AS()}")
print(f"Motor Octane Number (MON): {model.MON()}")
print(f"External Isobutane-to-Olefin Ratio (EIO): {model.EIO()}")
```

4. Input Problem Data and Test Model Code

```
# Installing pyomo nd solver
!pip install -q pyomo
!wget -N -q "https://matematica.unipv.it/gualandi/solvers/ipopt-linux64.zip"
!unzip -o -q ipopt-linux64
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 46.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.6/49.6 kB 3.1 MB/s eta 0:00:00
```

```
from pyomo.environ import *

# Create the model
model = ConcreteModel()

# Define the parameters (using sample data)
model.R = Param(initialize=0.63)  # Revenue per barrel of alkylate yield per unit of motor octane number
model.C_OF = Param(initialize=5.04)  # Cost per barrel of olefin feed
model.C_IR = Param(initialize=0.035)  # Cost per barrel of isobutane recycle
model.C_AA = Param(initialize=10.0)  # Cost per thousand pounds of acid addition rate
model.C_IM = Param(initialize=3.36)  # Cost per barrel of isobutane makeup

# Define the decision variables
model.OF = Var(domain=NonNegativeReals)  # Olefin feed (barrels per day)
model.IR = Var(domain=NonNegativeReals)  # Isobutane recycle (barrels per day)
model.AA = Var(domain=NonNegativeReals)  # Acid addition rate (thousands of pounds per day)
model.AY = Var(domain=NonNegativeReals)  # Alkylate yield (barrels per day)
model.IM = Var(domain=NonNegativeReals)  # Isobutane makeup (barrels per day)
model.AS = Var(domain=NonNegativeReals)  # Acid strength (weight percent)
model.MON = Var(domain=NonNegativeReals)  # Motor octane number
model.EIO = Var(domain=NonNegativeReals)  # External isobutane-to-olefin ratio
model.ADF = Var(domain=NonNegativeReals)  # Acid dilution factor
model.F4 = Var(domain=NonNegativeReals)  # F-4 performance number

# Define the objective function
def profit(model):
    return model.R * model.AY * model.MON - (model.C_OF * model.OF + model.C_IR * model.IR +
                                             model.C_AA * model.AA + model.C_IM * model.IM)
model.profit = Objective(rule=profit, sense=maximize)

# Define the constraints
def alkylate_yield(model):
    return model.AY == model.OF * (1.12 + 0.13167 * model.EIO - 0.00667 * model.EIO**2)
model.alkylate_yield_eq = Constraint(rule=alkylate_yield)

def alkylate_yield_bounds(model):
    return (0.98 * model.AY, model.OF * (1.12 + 0.13167 * model.EIO - 0.00667 * model.EIO**2), 1.02 * model.AY)
model.alkylate_yield_bounds = Constraint(rule=alkylate_yield_bounds)

def motor_octane_number(model):
    return model.MON == 86.35 + 1.098 * model.EIO - 0.038 * model.EIO**2 + 0.325 * (model.AS - 89)
model.motor_octane_number_eq = Constraint(rule=motor_octane_number)

def motor_octane_number_bounds(model):
    return (0.99 * model.MON, 86.35 + 1.098 * model.EIO - 0.038 * model.EIO**2 + 0.325 * (model.AS - 89), 1.01 * model.MON)
model.motor_octane_number_bounds = Constraint(rule=motor_octane_number_bounds)

def acid_dilution_factor(model):
    return model.ADF == 35.82 - 0.222 * model.F4
model.acid_dilution_factor_eq = Constraint(rule=acid_dilution_factor)

def acid_dilution_factor_bounds(model):
    return (0.95 * model.ADF, 35.82 - 0.222 * model.F4, 1.05 * model.ADF)
model.acid_dilution_factor_bounds = Constraint(rule=acid_dilution_factor_bounds)

def f4_performance_number(model):
    return model.F4 == -133 + 3 * model.MON
model.f4_performance_number_eq = Constraint(rule=f4_performance_number)

def f4_performance_number_bounds(model):
    return (0.90 * model.F4, -133 + 3 * model.MON, 1.10 * model.F4)
model.f4_performance_number_bounds = Constraint(rule=f4_performance_number_bounds)

def external_isobutane_to_olefin_ratio(model):
    return model.EIO == (model.IR + model.IM) / model.OF
model.external_isobutane_to_olefin_ratio_eq = Constraint(rule=external_isobutane_to_olefin_ratio)

def acid_strength(model):
    return model.AS == (93000 * model.AA) / (model.AA * model.ADF + 1000 * model.AA)
model.acid_strength_eq = Constraint(rule=acid_strength)

def material_balance(model):
    return 1.22 * model.AY == model.OF + model.IM
model.material_balance_eq = Constraint(rule=material_balance)

# Define variable bounds (using sample data)
model.OF_min = Param(initialize=0)
model.OF_max = Param(initialize=2000)
model.IR_min = Param(initialize=0)
model.IR_max = Param(initialize=16000)
model.AA_min = Param(initialize=0)
model.AA_max = Param(initialize=120)
```

```
model.IM_min = Param(initialize=0)
model.IM_max = Param(initialize=2000)
model.AS_min = Param(initialize=85)
model.AS_max = Param(initialize=93)
model.MON_min = Param(initialize=90)
model.MON_max = Param(initialize=95)
model.EIO_min = Param(initialize=3)
model.EIO_max = Param(initialize=12)
model.ADF_min = Param(initialize=1.2)
model.ADF_max = Param(initialize=4)
model.F4_min = Param(initialize=145)
model.F4_max = Param(initialize=162)

model.OF_bounds = Constraint(rule=lambda model: (model.OF_min, model.OF, model.OF_max))
model.IR_bounds = Constraint(rule=lambda model: (model.IR_min, model.IR, model.IR_max))
model.AA_bounds = Constraint(rule=lambda model: (model.AA_min, model.AA, model.AA_max))
model.IM_bounds = Constraint(rule=lambda model: (model.IM_min, model.IM, model.IM_max))
model.AS_bounds = Constraint(rule=lambda model: (model.AS_min, model.AS, model.AS_max))
model.MON_bounds = Constraint(rule=lambda model: (model.MON_min, model.MON, model.MON_max))
model.EIO_bounds = Constraint(rule=lambda model: (model.EIO_min, model.EIO, model.EIO_max))
model.ADF_bounds = Constraint(rule=lambda model: (model.ADF_min, model.ADF, model.ADF_max))
model.F4_bounds = Constraint(rule=lambda model: (model.F4_min, model.F4, model.F4_max))

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

# Print the results
print(f"Objective Value (Profit): {model.profit()}")
print(f"Olefin Feed (OF): {model.OF()}")
print(f"Isobutane Recycle (IR): {model.IR()}")
print(f"Acid Addition Rate (AA): {model.AA()}")
print(f"Alkylate Yield (AY): {model.AY()}")
print(f"Isobutane Makeup (IM): {model.IM()}")
print(f"Acid Strength (AS): {model.AS()}")
print(f"Motor Octane Number (MON): {model.MON()}")
print(f"External Isobutane-to-Olefin Ratio (EIO): {model.EIO()}")
print(f"Acid Dilution Factor (ADF): {model.ADF()}")
print(f"F-4 Performance Number (F4): {model.F4()}")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-8966de52c96a> in <cell line: 108>()
    106 # Solve the optimization problem
    107 solver = SolverFactory('ipopt')
--> 108 results = solver.solve(model)
    109
    110 # 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 'alkylate_yield_bounds' 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 the model
model = ConcreteModel()

# Define the parameters (using sample data)
model.R = Param(initialize=0.63)  # Revenue per barrel of alkylate yield per unit of motor octane number
model.C_OF = Param(initialize=5.04)  # Cost per barrel of olefin feed
model.C_IR = Param(initialize=0.035)  # Cost per barrel of isobutane recycle
model.C_AA = Param(initialize=10.0)  # Cost per thousand pounds of acid addition rate
model.C_IM = Param(initialize=3.36)  # Cost per barrel of isobutane makeup

# Define the decision variables
model.OF = Var(domain=NonNegativeReals)  # Olefin feed (barrels per day)
model.IR = Var(domain=NonNegativeReals)  # Isobutane recycle (barrels per day)
model.AA = Var(domain=NonNegativeReals)  # Acid addition rate (thousands of pounds per day)
model.AY = Var(domain=NonNegativeReals)  # Alkylate yield (barrels per day)
model.IM = Var(domain=NonNegativeReals)  # Isobutane makeup (barrels per day)
```

```
model.AS = Var(domain=NonNegativeReals)  # Acid strength (weight percent)
model.MON = Var(domain=NonNegativeReals)  # Motor octane number
model.EIO = Var(domain=NonNegativeReals)  # External isobutane-to-olefin ratio
model.ADF = Var(domain=NonNegativeReals)  # Acid dilution factor
model.F4 = Var(domain=NonNegativeReals)  # F-4 performance number

# Define the objective function
def profit(model):
    return model.R * model.AY * model.MON - (model.C_OF * model.OF + model.C_IR * model.IR +
                                             model.C_AA * model.AA + model.C_IM * model.IM)
model.profit = Objective(rule=profit, sense=maximize)

# Define the constraints
def alkylate_yield(model):
    return model.AY == model.OF * (1.12 + 0.13167 * model.EIO - 0.00667 * model.EIO**2)
model.alkylate_yield_eq = Constraint(rule=alkylate_yield)

def alkylate_yield_bound_lower(model):
    return (0.98 * model.AY <= model.OF * (1.12 + 0.13167 * model.EIO - 0.00667 * model.EIO**2))
model.alkylate_yield_bound_lower = Constraint(rule=alkylate_yield_bound_lower)

def alkylate_yield_bound_upper(model):
    return (model.OF * (1.12 + 0.13167 * model.EIO - 0.00667 * model.EIO**2) <= 1.02 * model.AY)
model.alkylate_yield_bound_upper = Constraint(rule=alkylate_yield_bound_upper)

def motor_octane_number(model):
    return model.MON == 86.35 + 1.098 * model.EIO - 0.038 * model.EIO**2 + 0.325 * (model.AS - 89)
model.motor_octane_number_eq = Constraint(rule=motor_octane_number)

def motor_octane_number_bound_lower(model):
    return (0.99 * model.MON <= 86.35 + 1.098 * model.EIO - 0.038 * model.EIO**2 + 0.325 * (model.AS - 89))
model.motor_octane_number_bound_lower = Constraint(rule=motor_octane_number_bound_lower)

def motor_octane_number_bound_upper(model):
    return (86.35 + 1.098 * model.EIO - 0.038 * model.EIO**2 + 0.325 * (model.AS - 89) <= 1.01 * model.MON)
model.motor_octane_number_bound_upper = Constraint(rule=motor_octane_number_bound_upper)

def acid_dilution_factor(model):
    return model.ADF == 35.82 - 0.222 * model.F4
model.acid_dilution_factor_eq = Constraint(rule=acid_dilution_factor)

def acid_dilution_factor_bound_lower(model):
    return (0.95 * model.ADF <= 35.82 - 0.222 * model.F4)
model.acid_dilution_factor_bound_lower = Constraint(rule=acid_dilution_factor_bound_lower)

def acid_dilution_factor_bound_upper(model):
    return (35.82 - 0.222 * model.F4 <= 1.05 * model.ADF)
model.acid_dilution_factor_bound_upper = Constraint(rule=acid_dilution_factor_bound_upper)

def f4_performance_number(model):
    return model.F4 == -133 + 3 * model.MON
model.f4_performance_number_eq = Constraint(rule=f4_performance_number)

def f4_performance_number_bound_lower(model):
    return (0.90 * model.F4 <= -133 + 3 * model.MON)
model.f4_performance_number_bound_lower = Constraint(rule=f4_performance_number_bound_lower)

def f4_performance_number_bound_upper(model):
    return (-133 + 3 * model.MON <= 1.10 * model.F4)
model.f4_performance_number_bound_upper = Constraint(rule=f4_performance_number_bound_upper)

def external_isobutane_to_olefin_ratio(model):
    return model.EIO == (model.IR + model.IM) / model.OF
model.external_isobutane_to_olefin_ratio_eq = Constraint(rule=external_isobutane_to_olefin_ratio)

def acid_strength(model):
    return model.AS == (93000 * model.AA) / (model.AA * model.ADF + 1000 * model.AA)
model.acid_strength_eq = Constraint(rule=acid_strength)

def material_balance(model):
    return 1.22 * model.AY == model.OF + model.IM
model.material_balance_eq = Constraint(rule=material_balance)

# Define variable bounds (using sample data)
model.OF_min = Param(initialize=0)
model.OF_max = Param(initialize=2000)
model.IR_min = Param(initialize=0)
model.IR_max = Param(initialize=16000)
model.AA_min = Param(initialize=0)
model.AA_max = Param(initialize=120)
model.IM_min = Param(initialize=0)
model.IM_max = Param(initialize=2000)
model.AS_min = Param(initialize=85)
```

```
model.AS_max = Param(initialize=93)
model.MON_min = Param(initialize=90)
model.MON_max = Param(initialize=95)
model.EIO_min = Param(initialize=3)
model.EIO_max = Param(initialize=12)
model.ADF_min = Param(initialize=1.2)
model.ADF_max = Param(initialize=4)
model.F4_min = Param(initialize=145)
model.F4_max = Param(initialize=162)

model.OF_bounds = Constraint(rule=lambda model: (model.OF_min, model.OF, model.OF_max))
model.IR_bounds = Constraint(rule=lambda model: (model.IR_min, model.IR, model.IR_max))
model.AA_bounds = Constraint(rule=lambda model: (model.AA_min, model.AA, model.AA_max))
model.IM_bounds = Constraint(rule=lambda model: (model.IM_min, model.IM, model.IM_max))
model.AS_bounds = Constraint(rule=lambda model: (model.AS_min, model.AS, model.AS_max))
model.MON_bounds = Constraint(rule=lambda model: (model.MON_min, model.MON, model.MON_max))
model.EIO_bounds = Constraint(rule=lambda model: (model.EIO_min, model.EIO, model.EIO_max))
model.ADF_bounds = Constraint(rule=lambda model: (model.ADF_min, model.ADF, model.ADF_max))
model.F4_bounds = Constraint(rule=lambda model: (model.F4_min, model.F4, model.F4_max))

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

# Print the results
print(f"Objective Value (Profit): {model.profit()}")
print(f"Olefin Feed (OF): {model.OF()}")
print(f"Isobutane Recycle (IR): {model.IR()}")
print(f"Acid Addition Rate (AA): {model.AA()}")
print(f"Alkylate Yield (AY): {model.AY()}")
print(f"Isobutane Makeup (IM): {model.IM()}")
print(f"Acid Strength (AS): {model.AS()}")
print(f"Motor Octane Number (MON): {model.MON()}")
print(f"External Isobutane-to-Olefin Ratio (EIO): {model.EIO()}")
print(f"Acid Dilution Factor (ADF): {model.ADF()}")
print(f"F-4 Performance Number (F4): {model.F4()}")
```

```
Objective Value (Profit): 171952.10683655558
Olefin Feed (OF): 1945.5990069237341
Isobutane Recycle (IR): 9391.1099301561
Acid Addition Rate (AA): 0.0
Alkylate Yield (AY): 3234.0975630522175
Isobutane Makeup (IM): 2000.0000199999708
Acid Strength (AS): 92.66363098967906
Motor Octane Number (MON): 92.66666618333375
External Isobutane-to-Olefin Ratio (EIO): 5.854808678262547
Acid Dilution Factor (ADF): 3.630000321899722
F-4 Performance Number (F4): 144.99999855000124
```