## 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
```

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)
```

```
To formulate the variables for this optimization problem, we can define the following:

Let:
- x1 = olefin feed (barrels per day)
- x2 = isobutane recycle (barrels per day)
- x3 = acid addition rate (thousands of pounds per day)
- x4 = alkylate yield (barrels per day)
- x5 = isobutane makeup (barrels per day)
- x6 = acid strength (weight percent)
- x7 = motor octane number
- x8 = external isobutane-to-olefin ratio
- x9 = acid dilution factor
- x10 = F-4 performance number

These are the 10 variables that will be used in the optimization problem to maximize the daily profit of the alkylation
```

## 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)
```

```
Objective function:

Let:
- p1 = profit per barrel of alkylate yield per unit of motor octane number
- c1 = cost per barrel of olefin feed
- c2 = cost per barrel of isobutane recycle
- c3 = cost per thousand pounds of acid addition rate
- c4 = cost per barrel of isobutane makeup
```

```
Maximize:
  p1 * x4 * x7 - (c1 * x1 + c2 * x2 + c3 * x3 + c4 * x5)

Subject to the given constraints and variable relationships.

The objective is to maximize the profit, which is revenue (p1 * x4 * x7) minus costs (c1 * x1 + c2 * x2 + c3 * x3 + c4 *

The revenue term p1 * x4 * x7 represents the profit per barrel of alkylate yield (x4) per unit of motor octane number (x

The cost term c1 * x1 + c2 * x2 + c3 * x3 + c4 * x5 captures the operational costs associated with olefin feed (x1), iso

This objective function aims to maximize the daily profit considering the revenue generated and the operational costs in
```

## 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)
```

```
Based on the given information, here are the constraints for this optimization problem:

1. Lower and upper bounds for variables:
   - LB_i ≤ xi ≤ UB_i, for i = 1, 2, ..., 10
     where LB_i and UB_i are the lower and upper bounds for each variable xi.

2. Alkylate yield regression estimate constraint:
   - 0.98 (1.12x1 + 0.13167x1x8 - 0.00667x1x8^2) ≤ x4 ≤ 1.02 (1.12x1 + 0.13167x1x8 - 0.00667x1x8^2)

3. Motor octane number regression estimate constraint:
   - 0.99 (86.35 + 1.098x8 - 0.038x8^2 + 0.325(x6 - 89)) ≤ x7 ≤ 1.01 (86.35 + 1.098x8 - 0.038x8^2 + 0.325(x6 - 89))

4. Acid dilution factor regression estimate constraint:
   - 0.95 (35.82 - 0.222x10) ≤ x9 ≤ 1.05 (35.82 - 0.222x10)

5. F-4 performance number regression estimate constraint:
   - 0.90 (-133 + 3x7) ≤ x10 ≤ 1.10 (-133 + 3x7)

6. External isobutane-to-olefin ratio constraint:
   - x8 = (x2 + x5) / x1

7. Acid strength constraint:
   - x6 = (93000x3) / (x3x9 + 1000x3)

8. Material balance constraint:
   - 1.22x4 = x1 + x5

These constraints, along with the objective function you provided, form the complete mathematical optimization problem f
```

## 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 = ConcreteModel()

# Define the variables
model.x1 = Var(bounds=(0, None), doc="Olefin feed (barrels per day)")
model.x2 = Var(bounds=(0, None), doc="Isobutane recycle (barrels per day)")
model.x3 = Var(bounds=(0, None), doc="Acid addition rate (thousands of pounds per day)")
model.x4 = Var(bounds=(0, None), doc="Alkylate yield (barrels per day)")
model.x5 = Var(bounds=(0, None), doc="Isobutane makeup (barrels per day)")
model.x6 = Var(bounds=(0, 100), doc="Acid strength (weight percent)")
model.x7 = Var(bounds=(0, None), doc="Motor octane number")
model.x8 = Var(bounds=(0, None), doc="External isobutane-to-olefin ratio")
model.x9 = Var(bounds=(0, None), doc="Acid dilution factor")
model.x10 = Var(bounds=(0, None), doc="F-4 performance number")

# Define the objective function
model.profit = Objective(expr=p1 * model.x4 * model.x7 - (c1 * model.x1 + c2 * model.x2 + c3 * model.x3 + c4 * model.x

# Define the constraints
model.alkylate_yield_lower = Constraint(expr=0.98 * (1.12 * model.x1 + 0.13167 * model.x1 * model.x8 - 0.00667 * model
model.alkylate_yield_upper = Constraint(expr=model.x4 <= 1.02 * (1.12 * model.x1 + 0.13167 * model.x1 * model.x8 - 0.0

model.octane_number_lower = Constraint(expr=0.99 * (86.35 + 1.098 * model.x8 - 0.038 * model.x8**2 + 0.325 * (model.x6
model.octane_number_upper = Constraint(expr=model.x7 <= 1.01 * (86.35 + 1.098 * model.x8 - 0.038 * model.x8**2 + 0.325

model.acid_dilution_lower = Constraint(expr=0.95 * (35.82 - 0.222 * model.x10) <= model.x9)
model.acid_dilution_upper = Constraint(expr=model.x9 <= 1.05 * (35.82 - 0.222 * model.x10))

model.performance_number_lower = Constraint(expr=0.90 * (-133 + 3 * model.x7) <= model.x10)
model.performance_number_upper = Constraint(expr=model.x10 <= 1.10 * (-133 + 3 * model.x7))

model.isobutane_olefin_ratio = Constraint(expr=model.x8 == (model.x2 + model.x5) / model.x1)
model.acid_strength = Constraint(expr=model.x6 == (93000 * model.x3) / (model.x3 * model.x9 + 1000 * model.x3))
model.material_balance = Constraint(expr=1.22 * model.x4 == model.x1 + model.x5)

# Solve the optimization problem
solver = SolverFactory('glpk')  # Specify the solver you want to use (e.g., 'glpk', 'cplex', 'gurobi')
results = solver.solve(model)

# Print the results
print("Optimization Results:")
print("Objective Function Value:", model.profit())
print("Variable Values:")
print("x1 (Olefin feed):", model.x1())
print("x2 (Isobutane recycle):", model.x2())
print("x3 (Acid addition rate):", model.x3())
print("x4 (Alkylate yield):", model.x4())
print("x5 (Isobutane makeup):", model.x5())
print("x6 (Acid strength):", model.x6())
print("x7 (Motor octane number):", model.x7())
print("x8 (External isobutane-to-olefin ratio):", model.x8())
print("x9 (Acid dilution factor):", model.x9())
print("x10 (F-4 performance number):", model.x10())
'''

Note: In this code, you need to provide values for the coefficients `p1`, `c1`, `c2`, `c3`, and `c4` in the objective

Also, make sure you have the Pyomo library installed (`pip install pyomo`) and have a solver available (e.g., GLPK, CP
```

## 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 the variables
model.x1 = Var(bounds=(0, 2000), doc="Olefin feed (barrels per day)")
model.x2 = Var(bounds=(0, 16000), doc="Isobutane recycle (barrels per day)")
model.x3 = Var(bounds=(0, 120), doc="Acid addition rate (thousands of pounds per day)")
model.x4 = Var(bounds=(0, 5000), doc="Alkylate yield (barrels per day)")
model.x5 = Var(bounds=(0, 2000), doc="Isobutane makeup (barrels per day)")
model.x6 = Var(bounds=(85, 93), doc="Acid strength (weight percent)")
model.x7 = Var(bounds=(90, 95), doc="Motor octane number")
model.x8 = Var(bounds=(3, 12), doc="External isobutane-to-olefin ratio")
model.x9 = Var(bounds=(1.2, 4), doc="Acid dilution factor")
model.x10 = Var(bounds=(145, 162), doc="F-4 performance number")

# Costs
p1, c1, c2, c3, c4 = 0.63, 5.04, 0.035, 10.0, 3.36

# Define the objective function
model.profit = Objective(expr=p1 * model.x4 * model.x7 - (c1 * model.x1 + c2 * model.x2 + c3 * model.x3 + c4 * model.x5), sen

# Define the constraints
model.alkylate_yield_lower = Constraint(expr=0.98 * (1.12 * model.x1 + 0.13167 * model.x1 * model.x8 - 0.00667 * model.x1 * m
model.alkylate_yield_upper = Constraint(expr=model.x4 <= 1.02 * (1.12 * model.x1 + 0.13167 * model.x1 * model.x8 - 0.00667 *

model.octane_number_lower = Constraint(expr=0.99 * (86.35 + 1.098 * model.x8 - 0.038 * model.x8**2 + 0.325 * (model.x6 - 89))
model.octane_number_upper = Constraint(expr=model.x7 <= 1.01 * (86.35 + 1.098 * model.x8 - 0.038 * model.x8**2 + 0.325 * (mod

model.acid_dilution_lower = Constraint(expr=0.95 * (35.82 - 0.222 * model.x10) <= model.x9)
model.acid_dilution_upper = Constraint(expr=model.x9 <= 1.05 * (35.82 - 0.222 * model.x10))

model.performance_number_lower = Constraint(expr=0.90 * (-133 + 3 * model.x7) <= model.x10)
model.performance_number_upper = Constraint(expr=model.x10 <= 1.10 * (-133 + 3 * model.x7))

model.isobutane_olefin_ratio = Constraint(expr=model.x8 == (model.x2 + model.x5) / model.x1)
model.acid_strength = Constraint(expr=model.x6 == (93000 * model.x3) / (model.x3 * model.x9 + 1000 * model.x3))
model.material_balance = Constraint(expr=1.22 * model.x4 == model.x1 + model.x5)

# Solve the optimization problem
solver = SolverFactory('glpk')  # Specify the solver you want to use (e.g., 'glpk', 'cplex', 'gurobi')
results = solver.solve(model)

# Print the results
print("Optimization Results:")
print("Objective Function Value:", model.profit())
print("Variable Values:")
print("x1 (Olefin feed):", model.x1())
print("x2 (Isobutane recycle):", model.x2())
print("x3 (Acid addition rate):", model.x3())
print("x4 (Alkylate yield):", model.x4())
print("x5 (Isobutane makeup):", model.x5())
print("x6 (Acid strength):", model.x6())
print("x7 (Motor octane number):", model.x7())
print("x8 (External isobutane-to-olefin ratio):", model.x8())
print("x9 (Acid dilution factor):", model.x9())
print("x10 (F-4 performance number):", model.x10())
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-61f16ef2abe2> in <cell line: 43>()
     41 # Solve the optimization problem
     42 solver = SolverFactory('glpk')  # Specify the solver you want to use
(e.g., 'glpk', 'cplex', 'gurobi')
---> 43 results = solver.solve(model)
     44 
     45 # Print the results

                                  9 frames

/usr/local/lib/python3.10/dist-packages/pyomo/repn/plugins/lp_writer.py in
write(self, model)
    376         repn = objective_visitor.walk_expression(obj.expr)
    377         if repn.nonlinear is not None:
--> 378             raise ValueError(
    379                 f"Model objective ({obj.name}) contains nonlinear
terms that "
    380                 "cannot be written to LP format"
```

## 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 the variables
model.x1 = Var(bounds=(0, 2000), doc="Olefin feed (barrels per day)")
model.x2 = Var(bounds=(0, 16000), doc="Isobutane recycle (barrels per day)")
model.x3 = Var(bounds=(0, 120), doc="Acid addition rate (thousands of pounds per day)")
model.x4 = Var(bounds=(0, 5000), doc="Alkylate yield (barrels per day)")
model.x5 = Var(bounds=(0, 2000), doc="Isobutane makeup (barrels per day)")
model.x6 = Var(bounds=(85, 93), doc="Acid strength (weight percent)")
model.x7 = Var(bounds=(90, 95), doc="Motor octane number")
model.x8 = Var(bounds=(3, 12), doc="External isobutane-to-olefin ratio")
model.x9 = Var(bounds=(1.2, 4), doc="Acid dilution factor")
model.x10 = Var(bounds=(145, 162), doc="F-4 performance number")

# Costs
p1, c1, c2, c3, c4 = 0.63, 5.04, 0.035, 10.0, 3.36

# Define the objective function
model.profit = Objective(expr=p1 * model.x4 * model.x7 - (c1 * model.x1 + c2 * model.x2 + c3 * model.x3 + c4 * model.x5), sen

# Define the constraints
model.alkylate_yield_lower = Constraint(expr=0.98 * (1.12 * model.x1 + 0.13167 * model.x1 * model.x8 - 0.00667 * model.x1 * m
model.alkylate_yield_upper = Constraint(expr=model.x4 <= 1.02 * (1.12 * model.x1 + 0.13167 * model.x1 * model.x8 - 0.00667 *

model.octane_number_lower = Constraint(expr=0.99 * (86.35 + 1.098 * model.x8 - 0.038 * model.x8**2 + 0.325 * (model.x6 - 89))
model.octane_number_upper = Constraint(expr=model.x7 <= 1.01 * (86.35 + 1.098 * model.x8 - 0.038 * model.x8**2 + 0.325 * (mod

model.acid_dilution_lower = Constraint(expr=0.95 * (35.82 - 0.222 * model.x10) <= model.x9)
model.acid_dilution_upper = Constraint(expr=model.x9 <= 1.05 * (35.82 - 0.222 * model.x10))

model.performance_number_lower = Constraint(expr=0.90 * (-133 + 3 * model.x7) <= model.x10)
model.performance_number_upper = Constraint(expr=model.x10 <= 1.10 * (-133 + 3 * model.x7))

model.isobutane_olefin_ratio = Constraint(expr=model.x8 == (model.x2 + model.x5) / model.x1)
model.acid_strength = Constraint(expr=model.x6 == (93000 * model.x3) / (model.x3 * model.x9 + 1000 * model.x3))
model.material_balance = Constraint(expr=1.22 * model.x4 == model.x1 + model.x5)

# Solve the optimization problem
solver = SolverFactory('ipopt')  # Specify the solver you want to use (e.g., 'glpk', 'cplex', 'gurobi')
results = solver.solve(model)

# Print the results
print("Optimization Results:")
print("Objective Function Value:", model.profit())
print("Variable Values:")
print("x1 (Olefin feed):", model.x1())
print("x2 (Isobutane recycle):", model.x2())
print("x3 (Acid addition rate):", model.x3())
print("x4 (Alkylate yield):", model.x4())
print("x5 (Isobutane makeup):", model.x5())
print("x6 (Acid strength):", model.x6())
print("x7 (Motor octane number):", model.x7())
print("x8 (External isobutane-to-olefin ratio):", model.x8())
print("x9 (Acid dilution factor):", model.x9())
print("x10 (F-4 performance number):", model.x10())
```

```
Optimization Results:
Objective Function Value: 176594.13705361064
Variable Values:
x1 (Olefin feed): 2000.0
x2 (Isobutane recycle): 10113.175625405223
x3 (Acid addition rate): 0.0
x4 (Alkylate yield): 3278.688557376949
x5 (Isobutane makeup): 2000.0
x6 (Acid strength): 92.88853376059824
x7 (Motor octane number): 93.79868152416277
x8 (External isobutane-to-olefin ratio): 6.056587762136942
x9 (Acid dilution factor): 1.2
x10 (F-4 performance number): 155.9263365300636
```