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 16.8 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 12.2 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 34.2 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 11.5 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 7.2 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)
```

```
Based on the problem description, let's define the variables for this mathematical optimization problem:

    Decision Variables:
    - OF: Olefin feed (barrels per day)
    - IR: Isobutane recycle (barrels per day)
    - AA: Acid addition rate (thousands of pounds per day)
    - IM: Isobutane makeup (barrels per day)

    Dependent Variables:
    - AY: Alkylate yield (barrels per day)
    - AS: Acid strength (weight percent)
```

```
- MO: Motor octane number
- ER: External isobutane-to-olefin ratio
- AD: Acid dilution factor
- FP: F-4 performance number

Objective Function:
- P: Daily profit (to be maximized)

The dependent variables (AY, AS, MO, ER, AD, and FP) will be expressed as functions of the decision variables using the
```

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)
```

```
Now, let's formulate the objective function for this alkylation process optimization problem:

Maximize:  P = (AY * MO) - (OF + IR + AA + IM)

Where:
- P is the daily profit to be maximized
- AY is the alkylate yield (barrels per day), expressed as a function of OF and ER
- MO is the motor octane number, expressed as a function of ER and AS
- OF is the olefin feed (barrels per day)
- IR is the isobutane recycle (barrels per day)
- AA is the acid addition rate (thousands of pounds per day)
- IM is the isobutane makeup (barrels per day)

The objective function aims to maximize the daily profit (P), which is calculated as the revenue generated from the prod

Please note that the dependent variables AY and MO are expressed as functions of the decision variables and other depend
```

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 provided problem description, here are the constraints for the alkylation process optimization problem:

Regression Formula Constraints:
1. AY = OF * (1.12 + 0.13167 * ER - 0.00667 * ER^2)
   0.98 * AY ≤ OF * (1.12 + 0.13167 * ER - 0.00667 * ER^2) ≤ 1.02 * AY

2. MO = 86.35 + 1.098 * ER - 0.038 * ER^2 + 0.325 * (AS - 89)
   0.99 * MO ≤ 86.35 + 1.098 * ER - 0.038 * ER^2 + 0.325 * (AS - 89) ≤ 1.01 * MO

3. AD = 35.82 - 0.222 * FP
   0.95 * AD ≤ 35.82 - 0.222 * FP ≤ 1.05 * AD

4. FP = -133 + 3 * MO
   0.9 * FP ≤ -133 + 3 * MO ≤ 1.1 * FP

Additional Constraints:
5. ER = (IR + IM) / OF
```

```
6. AS = (93000 * AA) / (AA * AD + 1000 * AA)

7. 1.22 * AY = OF + IM

Variable Bounds:
8. OF_min ≤ OF ≤ OF_max
9. IR_min ≤ IR ≤ IR_max
10. AA_min ≤ AA ≤ AA_max
11. IM_min ≤ IM ≤ IM_max

Where:
- OF_min, OF_max, IR_min, IR_max, AA_min, AA_max, IM_min, and IM_max are the respective lower and upper bounds for the d

The regression formula constraints (1-4) ensure that the dependent variables (AY, MO, AD, and FP) are within the specifi

The additional constraints (5-7) represent the relationships between variables based on the nature of the chemical proce

The variable bounds (8-11) specify the lower and upper limits for each decision variable.

These constraints, along with the objective function provided earlier, complete the formulation of the alkylation proces
```

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)
```

```
    # Define objective function
    def profit(model):
        return (model.AY * model.MO) - (model.OF + model.IR + model.AA + model.IM)
    model.profit = Objective(rule=profit, sense=maximize)

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

    def motor_octane(model):
        return model.MO == 86.35 + 1.098 * model.ER - 0.038 * model.ER**2 + 0.325 * (model.AS - 89)
```

```
solver = SolverFactory('ipopt')
results = solver.solve(model)

# Print the results
print("Optimization Results:")
print("Objective Function Value (Daily Profit): $", model.profit())
print("Olefin Feed (OF):", model.OF())
print("Isobutane Recycle (IR):", model.IR())
print("Acid Addition Rate (AA):", model.AA())
print("Isobutane Makeup (IM):", model.IM())
print("Alkylate Yield (AY):", model.AY())
print("Acid Strength (AS):", model.AS())
print("Motor Octane Number (MO):", model.MO())
print("External Isobutane-to-Olefin Ratio (ER):", model.ER())
print("Acid Dilution Factor (AD):", model.AD())
print("F-4 Performance Number (FP):", model.FP())
```

In this Pyomo model:
- Decision variables (OF, IR, AA, IM) and dependent variables (AY, AS, MO, ER, AD, FP) are defined.
- The objective function (profit) is defined to maximize the daily profit.
- Constraints are defined based on the regression formulas and additional relationships provided.
- The model is solved using the 'ipopt' solver (you may need to install it separately).

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 Pyomo model
model = ConcreteModel()

# Define decision variables
model.OF = Var(within=NonNegativeReals, bounds=(0, 2000))  # Sample bounds for OF
model.IR = Var(within=NonNegativeReals, bounds=(0, 16000))  # Sample bounds for IR
model.AA = Var(within=NonNegativeReals, bounds=(0, 120))  # Sample bounds for AA
model.IM = Var(within=NonNegativeReals, bounds=(0, 2000))  # Sample bounds for IM

# Define dependent variables
model.AY = Var(within=NonNegativeReals)
model.AS = Var(within=NonNegativeReals)
model.MO = Var(within=NonNegativeReals)
model.ER = Var(within=NonNegativeReals)
model.AD = Var(within=NonNegativeReals)
model.FP = Var(within=NonNegativeReals)

# Define objective function
def profit(model):
    return (model.AY * model.MO) - (model.OF + model.IR + model.AA + model.IM)
model.profit = Objective(rule=profit, sense=maximize)

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

def motor_octane(model):
    return model.MO == 86.35 + 1.098 * model.ER - 0.038 * model.ER**2 + 0.325 * (model.AS - 89)
model.motor_octane = Constraint(rule=motor_octane)

def acid_dilution(model):
    return model.AD == 35.82 - 0.222 * model.FP
model.acid_dilution = Constraint(rule=acid_dilution)

def f4_performance(model):
    return model.FP == -133 + 3 * model.MO
model.f4_performance = Constraint(rule=f4_performance)

def external_ratio(model):
```

```
def external_ratio(model):
    return model.ER == (model.IR + model.IM) / model.OF
model.external_ratio = Constraint(rule=external_ratio)

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

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

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

# Print the results
print("Optimization Results:")
print("Objective Function Value (Daily Profit): $", model.profit())
print("Olefin Feed (OF):", model.OF())
print("Isobutane Recycle (IR):", model.IR())
print("Acid Addition Rate (AA):", model.AA())
print("Isobutane Makeup (IM):", model.IM())
print("Alkylate Yield (AY):", model.AY())
print("Acid Strength (AS):", model.AS())
print("Motor Octane Number (MO):", model.MO())
print("External Isobutane-to-Olefin Ratio (ER):", model.ER())
print("Acid Dilution Factor (AD):", model.AD())
print("F-4 Performance Number (FP):", model.FP())
```

```
Optimization Results:
Objective Function Value (Daily Profit): $ 290019.20820197405
Olefin Feed (OF): 2000.0
Isobutane Recycle (IR): 8895.199970789998
Acid Addition Rate (AA): 0.0
Isobutane Makeup (IM): 2000.0
Alkylate Yield (AY): 3278.6885573769923
Acid Strength (AS): 92.64655368644925
Motor Octane Number (MO): 92.38889356880571
External Isobutane-to-Olefin Ratio (ER): 5.447599940919084
Acid Dilution Factor (AD): 3.8149968831754033
F-4 Performance Number (FP): 144.1666807064171
```

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

```
##### OMITTED BECAUSE CODE AND MODeL ARE EQUIVALENT (not exactly translated, but outcome will be the same) ######
```