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/MIP/MIP1.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:
 You are a city planner, looking to open facilities at some locations. We have a set of customers and a set of possible
The goal is to minimize the overall costs, which include both the fixed activation costs for any opened facilities and t
Please formulate this as a mathematical optimization model.
```

```
!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)
```

```
Great! Let's formulate this facility location problem as a mathematical optimization model. We'll start by defining the

Sets:
- I: set of customers, indexed by i
- J: set of potential facility locations, indexed by j

Parameters:
- d_i: demand of customer i
- f_j: fixed annual activation cost for opening a facility at location j
- c_ij: transportation cost for servicing customer i from facility j
- M_j: maximum annual service volume at facility j

Decision Variables:
- x_ij: fraction of customer i's demand serviced by facility j (continuous, 0 ≤ x_ij ≤ 1)
- y_j: binary variable indicating if a facility is opened at location j (y_j = 1 if opened, 0 otherwise)

Objective Function:
minimize ∑_j f_j * y_j + ∑_i ∑_j c_ij * d_i * x_ij

Constraints:
1. Each customer's demand must be fully met:
   ∑_j x_ij = 1, ∀i ∈ I

2. The service volume at each facility should not exceed its maximum annual limit:
   ∑_i d_i * x_ij ≤ M_j * y_j, ∀j ∈ J

3. Customers can only be serviced by opened facilities:
   x_ij ≤ y_j, ∀i ∈ I, ∀j ∈ J

4. Non-negativity and binary constraints:
   0 ≤ x_ij ≤ 1, ∀i ∈ I, ∀j ∈ J
   y_j ∈ {0, 1}, ∀j ∈ J

The objective function minimizes the total cost, which includes the fixed activation costs for opened facilities and the

This mathematical optimization model can be solved using mixed-integer linear programming (MILP) techniques to determine
```

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 ∑_j f_j * y_j + ∑_i ∑_j c_ij * d_i * x_ij

where:
- f_j is the fixed annual activation cost for opening a facility at location j
- y_j is a binary variable indicating if a facility is opened at location j (y_j = 1 if opened, 0 otherwise)
- c_ij is the transportation cost for servicing customer i from facility j
- d_i is the demand of customer i
- x_ij is a continuous variable representing the fraction of customer i's demand serviced by facility j (0 ≤ x_ij ≤ 1)

The objective is to minimize the total cost, which consists of two components:
1. ∑_j f_j * y_j : The sum of fixed activation costs for all opened facilities
2. ∑_i ∑_j c_ij * d_i * x_ij : The total transportation costs for servicing all customers from the opened facilities
```

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 given mathematical optimization problem:

1. Each customer's demand must be fully met:
   ∑_j x_ij = 1, ∀i ∈ I

   This constraint ensures that the sum of the fractions of demand serviced by all facilities for each customer i equals

2. The service volume at each facility should not exceed its maximum annual limit:
   ∑_i d_i * x_ij ≤ M_j * y_j, ∀j ∈ J

   This constraint guarantees that for each facility j, the total demand serviced (∑_i d_i * x_ij) does not exceed the m

3. Customers can only be serviced by opened facilities:
   x_ij ≤ y_j, ∀i ∈ I, ∀j ∈ J

   This constraint ensures that a customer i can only be serviced by a facility j if that facility is opened (y_j = 1).

4. Non-negativity and binary constraints:
   0 ≤ x_ij ≤ 1, ∀i ∈ I, ∀j ∈ J
   y_j ∈ {0, 1}, ∀j ∈ J

   These constraints define the range of values for the decision variables. The fraction of demand serviced (x_ij) must

These constraints, along with the objective function, complete the formulation of the mathematical optimization model fo
```

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.

```
response_p11_adjusted = """
Great! Let's formulate this facility location problem as a mathematical optimization model. We'll start by defining the sets

Sets:
- I: set of customers, indexed by i
- J: set of potential facility locations, indexed by j

Parameters:
- d_i: demand of customer i
- f_j: fixed annual activation cost for opening a facility at location j
- c_ij: transportation cost for servicing customer i from facility j
- M_j: maximum annual service volume at facility j

Decision Variables:
- x_ij: fraction of customer i's demand serviced by facility j (continuous, 0 ≤ x_ij ≤ 1)
- y_j: binary variable indicating if a facility is opened at location j (y_j = 1 if opened, 0 otherwise)
"""


message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": (response_p11_adjusted + response_p12 + response_p13)}
    ]
)

response_p2 = message2.content[0].text


# Print response
print(response_p2)
```

Here's a Python code using Pyomo to solve the facility location problem:

```python
from pyomo.environ import *

# Create a Pyomo model
model = ConcreteModel()

# Define sets
model.I = Set(initialize=['Customer1', 'Customer2', 'Customer3'])  # Sample data
model.J = Set(initialize=['Facility1', 'Facility2', 'Facility3'])  # Sample data

# Define parameters
model.d = Param(model.I, initialize={'Customer1': 100, 'Customer2': 150, 'Customer3': 200})  # Sample data
model.f = Param(model.J, initialize={'Facility1': 1000, 'Facility2': 1500, 'Facility3': 1200})  # Sample data
model.c = Param(model.I, model.J, initialize={
    ('Customer1', 'Facility1'): 10, ('Customer1', 'Facility2'): 12, ('Customer1', 'Facility3'): 15,
    ('Customer2', 'Facility1'): 8, ('Customer2', 'Facility2'): 9, ('Customer2', 'Facility3'): 11,
    ('Customer3', 'Facility1'): 7, ('Customer3', 'Facility2'): 10, ('Customer3', 'Facility3'): 13
})  # Sample data
model.M = Param(model.J, initialize={'Facility1': 500, 'Facility2': 600, 'Facility3': 400})  # Sample data

# Define decision variables
model.x = Var(model.I, model.J, bounds=(0, 1))
model.y = Var(model.J, within=Binary)

# Define objective function
def obj_rule(model):
    return sum(model.f[j] * model.y[j] for j in model.J) + \
           sum(model.c[i, j] * model.d[i] * model.x[i, j] for i in model.I for j in model.J)
model.obj = Objective(rule=obj_rule, sense=minimize)

# Define constraints
def demand_constraint_rule(model, i):
    return sum(model.x[i, j] for j in model.J) == 1
model.demand_constraint = Constraint(model.I, rule=demand_constraint_rule)

def capacity_constraint_rule(model, j):
    return sum(model.d[i] * model.x[i, j] for i in model.I) <= model.M[j] * model.y[j]
model.capacity_constraint = Constraint(model.J, rule=capacity_constraint_rule)

def service_constraint_rule(model, i, j):
    return model.x[i, j] <= model.y[j]
model.service_constraint = Constraint(model.I, model.J, rule=service_constraint_rule)

# Solve the model
solver = SolverFactory('glpk')
results = solver.solve(model)

# Print the results
print("Objective value:", value(model.obj))
for j in model.J:
    if value(model.y[j]) > 0.5:
        print(f"Open facility at location {j}")
```

```
            for i in model.I:
                if value(model.x[i, j]) > 0:
                    print(f"  Serve {value(model.x[i, j]) * 100}% of demand from customer {i}")
    ```
```

## 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 sets
model.I = Set(initialize=['Customer1', 'Customer2', 'Customer3', 'Customer4', 'Customer5'])  # Sample data
model.J = Set(initialize=['Facility1', 'Facility2', 'Facility3'])  # Sample data

# Define parameters
model.d = Param(model.I, initialize={'Customer1': 80, 'Customer2': 270, 'Customer3': 250, 'Customer4': 160, 'Customer5': 180}
model.f = Param(model.J, initialize={'Facility1': 1000, 'Facility2': 1000, 'Facility3': 1000})  # Sample data
model.c = Param(model.I, model.J, initialize={
    ('Customer1', 'Facility1'): 4, ('Customer1', 'Facility2'): 6, ('Customer1', 'Facility3'): 9,
    ('Customer2', 'Facility1'): 5, ('Customer2', 'Facility2'): 4, ('Customer2', 'Facility3'): 7,
    ('Customer3', 'Facility1'): 6, ('Customer3', 'Facility2'): 3, ('Customer3', 'Facility3'): 4,
    ('Customer4', 'Facility1'): 8, ('Customer4', 'Facility2'): 5, ('Customer4', 'Facility3'): 3,
    ('Customer5', 'Facility1'): 10, ('Customer5', 'Facility2'): 8, ('Customer5', 'Facility3'): 4
})  # Sample data
model.M = Param(model.J, initialize={'Facility1': 500, 'Facility2': 500, 'Facility3': 500}  # Sample maximum annual service v
)  # Sample data

# Define decision variables
model.x = Var(model.I, model.J, bounds=(0, 1))
model.y = Var(model.J, within=Binary)

# Define objective function
def obj_rule(model):
    return sum(model.f[j] * model.y[j] for j in model.J) + \
           sum(model.c[i, j] * model.d[i] * model.x[i, j] for i in model.I for j in model.J)
model.obj = Objective(rule=obj_rule, sense=minimize)

# Define constraints
def demand_constraint_rule(model, i):
    return sum(model.x[i, j] for j in model.J) == 1
model.demand_constraint = Constraint(model.I, rule=demand_constraint_rule)

def capacity_constraint_rule(model, j):
    return sum(model.d[i] * model.x[i, j] for i in model.I) <= model.M[j] * model.y[j]
model.capacity_constraint = Constraint(model.J, rule=capacity_constraint_rule)

def service_constraint_rule(model, i, j):
    return model.x[i, j] <= model.y[j]
model.service_constraint = Constraint(model.I, model.J, rule=service_constraint_rule)

# Solve the model
solver = SolverFactory('glpk')
results = solver.solve(model)

# Print the results
print("Objective value:", value(model.obj))
for j in model.J:
    if value(model.y[j]) > 0.5:
        print(f"Open facility at location {j}")
        for i in model.I:
            if value(model.x[i, j]) > 0:
                print(f"  Serve {value(model.x[i, j]) * 100}% of demand from customer {i}")
```

```
Objective value: 5609.999999999998
Open facility at location Facility2
  Serve 100.0% of demand from customer Customer1
  Serve 99.9999999999999% of demand from customer Customer2
  Serve 60.0% of demand from customer Customer3
Open facility at location Facility3
  Serve 40.0% of demand from customer Customer3
```
```

```
Serve 100.0% of demand from customer Customer4
Serve 100.0% of demand from customer Customer5
```

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