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)
```

```
To formulate this problem as a mathematical optimization model, let's first define the sets, parameters, and decision va

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

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

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

    The mathematical optimization model can be formulated as follows:

    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. A facility's service volume cannot exceed its maximum capacity:
       ∑_i (d_i * x_ij) ≤ u_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 costs, which include the fixed activation costs for opened facilities and 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)
```

```
Objective Function:
Minimize ∑_j (f_j * y_j) + ∑_i ∑_j (c_ij * d_i * x_ij)
```

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)
```

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

2. A facility's service volume cannot exceed its maximum capacity:
   ∑_i (d_i * x_ij) ≤ u_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
```

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 = """
To formulate this problem as a mathematical optimization model, let's first define the sets, parameters, and decision variab

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

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

Decision Variables:
- x_ij: fraction of customer i's demand serviced by facility j (continuous variable, 0 ≤ x_ij ≤ 1)
- y_j: binary variable indicating whether 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)
```

```
model.I = Set(initialize=['C1', 'C2', 'C3', 'C4', 'C5'])  # Set of customers
model.J = Set(initialize=['F1', 'F2', 'F3'])  # Set of potential facility locations

# Define parameters
model.f = Param(model.J, initialize={'F1': 1000, 'F2': 1200, 'F3': 1500})  # Fixed activation costs (sample data)
model.c = Param(model.I, model.J, initialize={
    ('C1', 'F1'): 10, ('C1', 'F2'): 12, ('C1', 'F3'): 15,
    ('C2', 'F1'): 8, ('C2', 'F2'): 9, ('C2', 'F3'): 11,
    ('C3', 'F1'): 13, ('C3', 'F2'): 16, ('C3', 'F3'): 20,
    ('C4', 'F1'): 7, ('C4', 'F2'): 6, ('C4', 'F3'): 9,
    ('C5', 'F1'): 14, ('C5', 'F2'): 17, ('C5', 'F3'): 22
})  # Transportation costs (sample data)
model.d = Param(model.I, initialize={'C1': 100, 'C2': 120, 'C3': 80, 'C4': 150, 'C5': 90})  # Customer demands (sample
model.u = Param(model.J, initialize={'F1': 400, 'F2': 500, 'F3': 600})  # Facility capacities (sample data)

# Define decision variables
model.x = Var(model.I, model.J, bounds=(0, 1))  # Fraction of customer demand serviced by facility
model.y = Var(model.J, within=Binary)  # Binary variable for facility opening

# 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.u[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 Function Value:", value(model.obj))
print("Facility Opening Decisions:")
for j in model.J:
    print(f"Facility {j}: {value(model.y[j])}")
print("Customer Assignment Decisions:")
for i in model.I:
    for j in model.J:
        if value(model.x[i, j]) > 0:
            print(f"Customer {i} assigned to Facility {j}: {value(model.x[i, j])}")
```

This code uses sample data for the fixed activation costs (`f`), transportation costs (`c`), customer demands (`d`), a

The code defines the sets, parameters, decision variables, objective function, and constraints according to the mathem

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=['C1', 'C2', 'C3', 'C4', 'C5'])  # Set of customers
model.J = Set(initialize=['F1', 'F2', 'F3'])  # Set of potential facility locations

# Define parameters
model.f = Param(model.J, initialize={'F1': 1000, 'F2': 1000, 'F3': 1000})  # Fixed activation costs (sample data)
```

```
model.c = Param(model.I, model.J, initialize={
    ('C1', 'F1'): 4, ('C1', 'F2'): 6, ('C1', 'F3'): 9,
    ('C2', 'F1'): 5, ('C2', 'F2'): 4, ('C2', 'F3'): 7,
    ('C3', 'F1'): 6, ('C3', 'F2'): 3, ('C3', 'F3'): 4,
    ('C4', 'F1'): 8, ('C4', 'F2'): 5, ('C4', 'F3'): 3,
    ('C5', 'F1'): 10, ('C5', 'F2'): 8, ('C5', 'F3'): 4
})  # Transportation costs (sample data)
model.d = Param(model.I, initialize={'C1': 80, 'C2': 270, 'C3': 250, 'C4': 160, 'C5': 180})  # Customer demands (sample data)
model.u = Param(model.J, initialize={'F1': 500, 'F2': 500, 'F3': 500})  # Facility capacities (sample data)

# Define decision variables
model.x = Var(model.I, model.J, bounds=(0, 1))  # Fraction of customer demand serviced by facility
model.y = Var(model.J, within=Binary)  # Binary variable for facility opening

# 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.u[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 Function Value:", value(model.obj))
print("Facility Opening Decisions:")
for j in model.J:
    print(f"Facility {j}: {value(model.y[j])}")
print("Customer Assignment Decisions:")
for i in model.I:
    for j in model.J:
        if value(model.x[i, j]) > 0:
            print(f"Customer {i} assigned to Facility {j}: {value(model.x[i, j])}")
```

```
Objective Function Value: 5609.999999999998
Facility Opening Decisions:
Facility F1: 0.0
Facility F2: 1.0
Facility F3: 1.0
Customer Assignment Decisions:
Customer C1 assigned to Facility F2: 1.0
Customer C2 assigned to Facility F1: 1.03361584812727e-15
Customer C2 assigned to Facility F2: 0.999999999999999
Customer C3 assigned to Facility F2: 0.6
Customer C3 assigned to Facility F3: 0.4
Customer C4 assigned to Facility F3: 1.0
Customer C5 assigned to Facility F3: 1.0
```

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