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 problem as a mathematical optimization model. We'll start by defining the sets, parameters,

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 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 a facility is opened, 0 otherwi

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 must 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 formulation represents the problem as a mixed-integer linear programming (MILP) model, which can be solved using a
```

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)
```

```
The objective function for this mathematical optimization problem is:

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 whether a facility is opened at location j (y_j = 1 if a facility is opened, 0 oth
- c_ij is the transportation cost for servicing customer i from facility j
- d_i is the demand of customer i
- x_ij is the fraction of customer i's demand serviced by facility j (continuous variable, 0 ≤ x_ij ≤ 1)

The objective function minimizes the total cost, which includes:
1. The sum of fixed activation costs for all opened facilities (∑_j f_j * y_j)
2. The sum of transportation costs for servicing all customers from all facilities (∑_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)
```

```
The constraints for this mathematical optimization problem are:

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 customer i's demand serviced by all facilities equals 1, mea

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

   This constraint ensures that the total demand serviced by facility j (∑_i d_i * x_ij) does not exceed its maximum ann

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 facility j (x_ij > 0) if that facility is opened (y

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

   These constraints define the bounds and binary restrictions for the decision variables. The fraction of customer i's
```

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 problem as a mathematical optimization model. We'll start by defining the sets, parameters, and

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 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 a facility is 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=['A', 'B', 'C', 'D', 'E'])  # Set of customers (sample data)
model.J = Set(initialize=['1', '2', '3', '4'])  # Set of potential facility locations (sample data)

# Define parameters
model.d = Param(model.I, initialize={'A': 100, 'B': 150, 'C': 200, 'D': 120, 'E': 180})  # Demand of customers (sample
model.f = Param(model.J, initialize={'1': 5000, '2': 6000, '3': 4500, '4': 5500})  # Fixed annual activation cost for
model.c = Param(model.I, model.J, initialize={
    ('A', '1'): 10, ('A', '2'): 12, ('A', '3'): 8, ('A', '4'): 11,
    ('B', '1'): 9, ('B', '2'): 10, ('B', '3'): 11, ('B', '4'): 13,
    ('C', '1'): 7, ('C', '2'): 8, ('C', '3'): 9, ('C', '4'): 10,
    ('D', '1'): 11, ('D', '2'): 13, ('D', '3'): 10, ('D', '4'): 12,
    ('E', '1'): 8, ('E', '2'): 9, ('E', '3'): 10, ('E', '4'): 11,
})  # Transportation cost for servicing customers from facilities (sample data)
model.m = Param(model.J, initialize={'1': 500, '2': 600, '3': 550, '4': 450})  # Maximum annual service volume for fac

# Define decision variables
model.x = Var(model.I, model.J, bounds=(0, 1))  # Fraction of customer's demand serviced by facility
model.y = Var(model.J, within=Binary)  # Binary variable indicating whether a facility is opened

# Define objective function
def objective_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
model.objective = Objective(rule=objective_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.objective))
print("Facility locations:")
for j in model.J:
```

In this code, we define the sets, parameters, decision variables, objective function, and constraints according to the

The `glpk` solver is used to solve the model, and the optimal solution is printed, including the objective value, the

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 *

# Define the model
model = ConcreteModel()

# Define sets
model.I = Set(initialize=['A', 'B', 'C', 'D', 'E'])  # Set of customers (sample data)
model.J = Set(initialize=['1', '2', '3'])  # Set of potential facility locations (sample data)

# Define parameters
model.d = Param(model.I, initialize={'A': 80, 'B': 270, 'C': 250, 'D': 160, 'E': 180})  # Demand of customers (sample data)
model.f = Param(model.J, initialize={'1': 1000, '2': 1000, '3': 1000})  # Fixed annual activation cost for facilities (sample
model.c = Param(model.I, model.J, initialize={
    ('A', '1'): 4, ('A', '2'): 6, ('A', '3'): 9,
    ('B', '1'): 5, ('B', '2'): 4, ('B', '3'): 7,
    ('C', '1'): 6, ('C', '2'): 3, ('C', '3'): 4,
    ('D', '1'): 8, ('D', '2'): 5, ('D', '3'): 3,
    ('E', '1'): 10, ('E', '2'): 8, ('E', '3'): 4
})  # Transportation cost for servicing customers from facilities (sample data)
model.m = Param(model.J, initialize={'1': 500, '2': 500, '3': 500})  # Maximum annual service volume for facilities (sample d

# Define decision variables
model.x = Var(model.I, model.J, bounds=(0, 1))  # Fraction of customer's demand serviced by facility
model.y = Var(model.J, within=Binary)  # Binary variable indicating whether a facility is opened

# Define objective function
def objective_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
model.objective = Objective(rule=objective_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.objective))
print("Facility locations:")
for j in model.J:
    if value(model.y[j]) > 0.5:
        print(f"Facility {j} is opened")
print("Customer assignments:")
for i in model.I:
    for j in model.J:
        if value(model.x[i,j]) > 0:
            print(f"Customer {i} is serviced by facility {j} with fraction {value(model.x[i,j])}")
```

```
Objective value: 5609.999999999998
Facility locations:
Facility 2 is opened
Facility 3 is opened
Customer assignments:
Customer A is serviced by facility 2 with fraction 1.0
```

```
Customer B is serviced by facility 1 with fraction 1.03361584812727e-15
Customer B is serviced by facility 2 with fraction 0.999999999999999
Customer C is serviced by facility 2 with fraction 0.6
Customer C is serviced by facility 3 with fraction 0.4
Customer D is serviced by facility 3 with fraction 1.0
Customer E is serviced by facility 3 with fraction 1.0
```

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