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
prompt1_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt1_MathematicalModel.txt'
prompt2_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt2_PyomoCode.txt'
problem_desc_path = '/content/drive/MyDrive/Thesis/ProblemDescriptions/MIP/MIP1.txt'

prompt1_file = open(prompt1_path, "r")
prompt2_file = open(prompt2_path, "r")
problem_desc_file = open(problem_desc_path, "r")

prompt1 = prompt1_file.read()
print("Prompt 1:\n", prompt1)

prompt2 = prompt2_file.read()
print("Prompt 2:\n", prompt2)

problem_desc = problem_desc_file.read()
print("Problem Description:\n", problem_desc)
```

```
Prompt 1:
 Please write a mathematical optimization model for this problem. Include parameters, decision variables, the objective
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: Create Mathematical Model

```
message = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt1,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)

response_p1 = message.content[0].text


# Print response
print(response_p1)
```

```
Great! Let's formulate this problem as a mathematical optimization model. We'll define the necessary sets, parameters, d

Sets:
- I: Set of customers
- J: Set of potential facility locations

Parameters:
- d_i: Demand of customer i ∈ I
- f_j: Fixed annual activation cost for facility j ∈ J
- c_ij: Transportation cost for servicing customer i ∈ I from facility j ∈ J
- M_j: Maximum annual service volume for facility j ∈ 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 facility j is opened (1) or not (0)

Objective Function:
Minimize total costs = Fixed activation costs + Transportation costs
min ∑_(j∈J) f_j * y_j + ∑_(i∈I) ∑_(j∈J) c_ij * d_i * x_ij

Constraints:
1. Each customer's demand must be fully met:
   ∑_(j∈J) x_ij = 1, ∀i ∈ I

2. Each facility's service volume cannot exceed its maximum annual limit:
   ∑_(i∈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 costs, which include 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 op
```

2. Prompt 2: Write Code based on Mathematical Model

```
message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": response_p1}
    ]
)

response_p2 = message2.content[0].text


# Print response
print(response_p2)
```

```
    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))
    print("Facility openings:")
    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]) > 0.001:
                print(f"Customer {i} is serviced by Facility {j}: {value(model.x[i, j]):.2f}")
    ```

    In this code, I have used sample data for the demand, fixed activation costs, maximum annual service volumes, and tran

    The code follows the mathematical formulation we discussed earlier. It creates sets for customers and facilities, defi
```

## 4. Input Problem Data and Test Model Code

```
# Installing pyomo nd solver
!pip install -q pyomo
!apt-get install -y -qq glpk-utils
!pip install glpk
```

Show hidden output

```
from pyomo.environ import *

# Sample data
customers = ['C1', 'C2', 'C3', 'C4', 'C5']
facilities = ['F1', 'F2', 'F3']

demand = {'C1': 80, 'C2': 270, 'C3': 250, 'C4': 160, 'C5': 180}  # Sample demand data
fixed_cost = {'F1': 1000, 'F2': 1000, 'F3': 1000}  # Sample fixed activation costs
max_capacity = {'F1': 500, 'F2': 500, 'F3': 500}  # Sample maximum annual service volumes

# Sample transportation costs (c_ij)
trans_cost = {
    ('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
}

# Create the Pyomo model
model = ConcreteModel()

# Define sets
model.I = Set(initialize=customers)
model.J = Set(initialize=facilities)

# Define parameters
model.d = Param(model.I, initialize=demand)
model.f = Param(model.J, initialize=fixed_cost)
```
```

```
model.c = Param(model.I, model.J, initialize=trans_cost)
model.M = Param(model.J, initialize=max_capacity)

# 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))
print("Facility openings:")
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]) > 0.001:
            print(f"Customer {i} is serviced by Facility {j}: {value(model.x[i, j]):.2f}")
```

```
Objective value: 5609.999999999998
Facility openings:
Facility F2 is opened
Facility F3 is opened
Customer assignments:
Customer C1 is serviced by Facility F2: 1.00
Customer C2 is serviced by Facility F2: 1.00
Customer C3 is serviced by Facility F2: 0.60
Customer C3 is serviced by Facility F3: 0.40
Customer C4 is serviced by Facility F3: 1.00
Customer C5 is serviced by Facility F3: 1.00
```

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