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
```

```
Collecting anthropic
  Downloading anthropic-0.26.0-py3-none-any.whl (877 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 877.7/877.7 kB 11.6 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 10.1 MB/s eta 0:00:00
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (2.7.1)
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 8.6 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 6.8 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==2.18.2 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: 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: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.
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: h11, httpcore, httpx, anthropic
Successfully installed anthropic-0.26.0 h11-0.14.0 httpcore-1.0.5 httpx-0.27.0
```

```
# 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)
```

```
Thank you for providing the problem description. I will formulate this problem as a mathematical optimization model, inc

Parameters:
- I: set of customers, indexed by i
- J: set of potential facility locations, indexed by j
- 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
- s_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 the total cost, which includes the fixed activation costs for opened facilities and the transportation costs fo

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

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

2. The total demand serviced by each facility cannot exceed its maximum service volume:
   ∑_i x_ij * d_i ≤ s_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

This mathematical optimization model represents the problem of minimizing the total cost of opening facilities and servi
```

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)
```

```
Here's a Python code using Pyomo to solve the facility location problem based on the provided mathematical formulation

```python
import pyomo.environ as pyo

# Define the model
model = pyo.ConcreteModel()

# Define the sets
model.I = pyo.Set(initialize=['A', 'B', 'C', 'D', 'E'])  # Sample customer locations
model.J = pyo.Set(initialize=['1', '2', '3'])  # Sample potential facility locations

# Define the parameters
model.f = pyo.Param(model.J, initialize={'1': 1000, '2': 1500, '3': 1200})  # Sample fixed costs
model.c = pyo.Param(model.I, model.J, initialize={
    ('A', '1'): 10, ('A', '2'): 12, ('A', '3'): 15,
    ('B', '1'): 8, ('B', '2'): 14, ('B', '3'): 11,
    ('C', '1'): 13, ('C', '2'): 9, ('C', '3'): 16,
    ('D', '1'): 7, ('D', '2'): 11, ('D', '3'): 10,
    ('E', '1'): 12, ('E', '2'): 10, ('E', '3'): 14
})  # Sample transportation costs
model.d = pyo.Param(model.I, initialize={'A': 100, 'B': 120, 'C': 80, 'D': 90, 'E': 110})  # Sample demands
model.s = pyo.Param(model.J, initialize={'1': 300, '2': 350, '3': 400})  # Sample service volumes

# Define the decision variables
model.x = pyo.Var(model.I, model.J, bounds=(0, 1))  # Fraction of demand serviced
model.y = pyo.Var(model.J, domain=pyo.Binary)  # Facility opening decision

# Define the objective function
model.obj = pyo.Objective(expr=sum(model.f[j] * model.y[j] for j in model.J) +
                          sum(model.c[i, j] * model.x[i, j] * model.d[i] for i in model.I for j in model.J),
                          sense=pyo.minimize)

# Define the constraints
model.demand_constraint = pyo.Constraint(model.I, rule=lambda model, i:
                                         sum(model.x[i, j] for j in model.J) == 1)
model.capacity_constraint = pyo.Constraint(model.J, rule=lambda model, j:
                                           sum(model.x[i, j] * model.d[i] for i in model.I) <= model.s[j] * model.y[j]
model.service_constraint = pyo.Constraint(model.I, model.J, rule=lambda model, i, j:
                                          model.x[i, j] <= model.y[j])

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

# Print the results
print("Optimal Solution:")
print("Objective Value =", pyo.value(model.obj))
for j in model.J:
    if pyo.value(model.y[j]) > 0.5:
        print(f"Facility {j} is opened")
        for i in model.I:
            if pyo.value(model.x[i, j]) > 0:
                print(f"  Customer {i} is serviced by Facility {j} with {pyo.value(model.x[i, j]) * 100}% of demand")
```

In this code, sample data is used for the following parameters:
```

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

```
import pyomo.environ as pyo
```

```
# Define the model
model = pyo.ConcreteModel()

# Define the sets
model.I = pyo.Set(initialize=['A', 'B', 'C', 'D', 'E'])  # Sample customer locations
model.J = pyo.Set(initialize=['1', '2', '3'])  # Sample potential facility locations

# Define the parameters
model.f = pyo.Param(model.J, initialize={'1': 1000, '2': 1000, '3': 1000})  # Sample fixed costs
model.c = pyo.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
})  # Sample transportation costs
model.d = pyo.Param(model.I, initialize={'A': 80, 'B': 270, 'C': 250, 'D': 160, 'E': 180})  # Sample demands
model.s = pyo.Param(model.J, initialize={'1': 500, '2': 500, '3': 500})  # Sample service volumes

# Define the decision variables
model.x = pyo.Var(model.I, model.J, bounds=(0, 1))  # Fraction of demand serviced
model.y = pyo.Var(model.J, domain=pyo.Binary)  # Facility opening decision

# Define the objective function
model.obj = pyo.Objective(expr=sum(model.f[j] * model.y[j] for j in model.J) +
                          sum(model.c[i, j] * model.x[i, j] * model.d[i] for i in model.I for j in model.J),
                          sense=pyo.minimize)

# Define the constraints
model.demand_constraint = pyo.Constraint(model.I, rule=lambda model, i:
                                         sum(model.x[i, j] for j in model.J) == 1)
model.capacity_constraint = pyo.Constraint(model.J, rule=lambda model, j:
                                           sum(model.x[i, j] * model.d[i] for i in model.I) <= model.s[j] * model.y[j])
model.service_constraint = pyo.Constraint(model.I, model.J, rule=lambda model, i, j:
                                          model.x[i, j] <= model.y[j])

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

# Print the results
print("Optimal Solution:")
print("Objective Value =", pyo.value(model.obj))
for j in model.J:
    if pyo.value(model.y[j]) > 0.5:
        print(f"Facility {j} is opened")
        for i in model.I:
            if pyo.value(model.x[i, j]) > 0:
                print(f"  Customer {i} is serviced by Facility {j} with {pyo.value(model.x[i, j]) * 100}% of demand")
```

```
Optimal Solution:
Objective Value = 5609.999999999998
Facility 2 is opened
  Customer A is serviced by Facility 2 with 100.0% of demand
  Customer B is serviced by Facility 2 with 99.9999999999999% of demand
  Customer C is serviced by Facility 2 with 60.0% of demand
Facility 3 is opened
  Customer C is serviced by Facility 3 with 40.0% of demand
  Customer D is serviced by Facility 3 with 100.0% of demand
  Customer E is serviced by Facility 3 with 100.0% of demand
```

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