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 5.4 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 7.9 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 8.3 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 5.5 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)
```

```
To formulate the mathematical optimization model, let's define the sets, parameters, decision variables, objective funct

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 facility j
- c_ij: transportation cost for servicing customer i from facility j
- M_j: maximum annual service volume for facility 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 (y_j = 1) or not (y_j = 0)

Objective Function:
Minimize the total costs, which include fixed activation costs and transportation costs:
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 complete mathematical optimization model is:

minimize ∑_j f_j * y_j + ∑_i ∑_j c_ij * d_i * x_ij
subject to:
∑_j x_ij = 1, ∀i ∈ I
∑_i d_i * x_ij ≤ M_j * y_j, ∀j ∈ J
x_ij ≤ y_j, ∀i ∈ I, ∀j ∈ J
0 ≤ x_ij ≤ 1, ∀i ∈ I, ∀j ∈ J
y_j ∈ {0, 1}, ∀j ∈ J

This model is a mixed-integer linear programming (MILP) problem, as it involves both continuous (x_ij) and binary (y_j)
```

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)
```

```
# Create a Pyomo model
model = ConcreteModel()

# Define sets
model.I = Set(initialize=['1', '2', '3', '4', '5'])  # Set of customers (sample data)
model.J = Set(initialize=['A', 'B', 'C'])  # Set of potential facility locations (sample data)

# Define parameters
model.d = Param(model.I, initialize={'1': 100, '2': 150, '3': 200, '4': 120, '5': 180})  # Demand of customers (sample
model.f = Param(model.J, initialize={'A': 5000, 'B': 6000, 'C': 4500})  # Fixed annual activation cost for facilities
model.c = Param(model.I, model.J, initialize={
    ('1', 'A'): 10, ('1', 'B'): 12, ('1', 'C'): 15,
    ('2', 'A'): 8, ('2', 'B'): 14, ('2', 'C'): 11,
    ('3', 'A'): 13, ('3', 'B'): 9, ('3', 'C'): 16,
    ('4', 'A'): 17, ('4', 'B'): 10, ('4', 'C'): 12,
    ('5', 'A'): 11, ('5', 'B'): 16, ('5', 'C'): 9
})  # Transportation cost for servicing customers from facilities (sample data)
model.M = Param(model.J, initialize={'A': 1000, 'B': 1200, 'C': 900})  # Maximum annual service volume for facilities

# Define decision variables
model.x = Var(model.I, model.J, bounds=(0, 1))  # Fraction of customer's demand serviced by a facility
model.y = Var(model.J, within=Binary)  # Binary variable indicating whether a facility is opened or not

# 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
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:", model.obj())
print("Facility locations:")
for j in model.J:
    if model.y[j].value == 1:
        print(f"Facility {j} is opened")
    else:
        print(f"Facility {j} is not opened")

print("Customer assignments:")
for i in model.I:
    for j in model.J:
        if model.x[i,j].value > 0:
```

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 *

# Create a Pyomo model
model = ConcreteModel()

# Define sets
model.I = Set(initialize=['1', '2', '3', '4', '5'])  # Set of customers (sample data)
model.J = Set(initialize=['A', 'B', 'C'])  # Set of potential facility locations (sample data)

# Define parameters160
model.d = Param(model.I, initialize={'1': 80, '2': 270, '3': 250, '4': 160, '5': 180})  # Demand of customers (sample data)
model.f = Param(model.J, initialize={'A': 1000, 'B': 1000, 'C': 1000})  # Fixed annual activation cost for facilities (sample
model.c = Param(model.I, model.J, initialize={
    ('1', 'A'): 4, ('1', 'B'): 6, ('1', 'C'): 9,
    ('2', 'A'): 5, ('2', 'B'): 4, ('2', 'C'): 7,
    ('3', 'A'): 6, ('3', 'B'): 3, ('3', 'C'): 4,
    ('4', 'A'): 8, ('4', 'B'): 5, ('4', 'C'): 3,
    ('5', 'A'): 10, ('5', 'B'): 8, ('5', 'C'): 4
})  # Transportation cost for servicing customers from facilities (sample data)
model.M = Param(model.J, initialize={'A': 500, 'B': 500, 'C': 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 a facility
model.y = Var(model.J, within=Binary)  # Binary variable indicating whether a facility is opened or not

# 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
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:", model.obj())
print("Facility locations:")
for j in model.J:
    if model.y[j].value == 1:
        print(f"Facility {j} is opened")
    else:
        print(f"Facility {j} is not opened")

print("Customer assignments:")
for i in model.I:
    for j in model.J:
        if model.x[i,j].value > 0:
            print(f"Customer {i} is serviced by facility {j} with {model.x[i,j].value:.2f} of its demand")
```

```
Objective value: 5609.999999999998
Facility locations:
Facility A is not opened
Facility B is opened
Facility C is opened
Customer assignments:
Customer 1 is serviced by facility B with 1.00 of its demand
Customer 2 is serviced by facility A with 0.00 of its demand
Customer 2 is serviced by facility B with 1.00 of its demand
Customer 3 is serviced by facility B with 0.60 of its demand
Customer 3 is serviced by facility C with 0.40 of its demand
Customer 4 is serviced by facility C with 1.00 of its demand
Customer 5 is serviced by facility C with 1.00 of its demand
```

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