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/MIP4.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 task with maximizing the water flow in a network of pipes over 10 time steps. The water enters the system at po

These are the pipe capacities:
AB: 3
AC: 6
AF: 1
BC: 3
BD: 10
BE: 4
CE: 4
CF: 4
DE: 5
EG: 5
FB: 12
FG: 7
```

```
!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)
```

```
Here is the mathematical optimization model for the given problem:

Parameters:
- T: Total number of time steps (T = 10)
- N: Total number of points (N = 7, labeled A to G)
- P: Total number of pipes (P = 12)
- K: Number of pipes that can be activated (K = 10)
- c_ij: Capacity of the pipe connecting points i and j (given in the problem)
- f(t): Water inflow at point A at time t, where f(t) = max(-4/30t^3 + t^2 - 0.234t + 3, 0)

Decision Variables:
- x_ij(t): Amount of water flowing from point i to point j at time t (continuous variable)
- y_ij: Binary variable indicating if the pipe connecting points i and j is activated (1 if activated, 0 otherwise)
- r_i(t): Amount of water released from point i at time t (continuous variable)

Objective Function:
Maximize the total amount of water flowing to point G over all time steps:
max ∑_t (x_EG(t) + x_FG(t))

Constraints:
1. Water balance at each point i and time t (except for point A and G):
   ∑_j x_ji(t-1) + r_i(t-1) = ∑_j x_ij(t) + r_i(t), ∀i ≠ A, G, ∀t > 1

2. Water inflow at point A at each time t:
   ∑_j x_Aj(t) + r_A(t) = f(t), ∀t

3. Pipe capacity constraints:
   x_ij(t) ≤ c_ij * y_ij, ∀i, j, ∀t

4. Limit on the number of activated pipes:
   ∑_i ∑_j y_ij ≤ K

5. Binary constraints on pipe activation variables:
   y_ij ∈ {0, 1}, ∀i, j

6. Non-negativity constraints:
   x_ij(t) ≥ 0, ∀i, j, ∀t
   r_i(t) ≥ 0, ∀i, ∀t

Note: The indices i and j in the constraints represent the points connected by the pipes, and the summation ∑_j is over

This optimization model aims to maximize the total amount of water flowing to point G over all time steps while satisfyi
```

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.T = pyo.RangeSet(1, T)
model.N = pyo.RangeSet(0, N-1)
model.P = [(i, j) for i in range(N) for j in range(N) if (i, j) in pipe_capacities]

# Decision variables
model.x = pyo.Var(model.P, model.T, domain=pyo.NonNegativeReals)
model.y = pyo.Var(model.P, domain=pyo.Binary)
model.r = pyo.Var(model.N, model.T, domain=pyo.NonNegativeReals)

# Objective function
model.obj = pyo.Objective(expr=sum(model.x[4, 6, t] + model.x[5, 6, t] for t in model.T), sense=pyo.maximize)

# Constraints
def water_balance(model, i, t):
    if i == 0 or i == 6:
        return pyo.Constraint.Skip
    inflow = sum(model.x[j, i, t-1] for j in model.N if (j, i) in model.P)
    outflow = sum(model.x[i, j, t] for j in model.N if (i, j) in model.P)
    return inflow + model.r[i, t-1] == outflow + model.r[i, t]

model.water_balance = pyo.Constraint(model.N, model.T, rule=water_balance)

def water_inflow(model, t):
    outflow = sum(model.x[0, j, t] for j in model.N if (0, j) in model.P)
    return outflow + model.r[0, t] == f(t)

model.water_inflow = pyo.Constraint(model.T, rule=water_inflow)

def pipe_capacity(model, i, j, t):
    return model.x[i, j, t] <= pipe_capacities[i, j] * model.y[i, j]

model.pipe_capacity = pyo.Constraint(model.P, model.T, rule=pipe_capacity)

model.pipe_activation = pyo.Constraint(expr=sum(model.y[i, j] for (i, j) in model.P) <= K)

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

# Print the results
print("Objective value:", pyo.value(model.obj))
print("Activated pipes:")
for (i, j) in model.P:
    if pyo.value(model.y[i, j]) > 0.5:
        print(f"Pipe from {i} to {j}")
```
```

In this code, I used sample data for the pipe capacities (`pipe_capacities`) since the actual values were not provid

The code follows the optimization model formulation closely. It defines the sets, decision variables, objective func

The `glpk` solver is used to solve the optimization problem. Finally, the objective value and the activated pipes ar

Please note that this code assumes that the indices of the points are from 0 to 6, with A being 0, B being 1, and so
```

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
!pip install pandas
```

Show hidden output

```
import pyomo.environ as pyo
import numpy as np

# Parameters
T = 10
N = 7
P = 12
K = 10

# Sample data for pipe capacities (c_ij)
# Replace these with the actual values from the problem
pipe_capacities = {
    (0, 1): 3, (0, 2): 6, (0, 5): 1, (1, 2): 3,
    (1, 3): 10, (1, 4): 4, (2, 4): 4, (2, 5): 4,
    (3, 4): 5, (4, 6): 5, (5, 1): 12, (5, 6): 7
}

# Water inflow function at point A
```

```
def f(t):
    return max(-4/30 * t**3 + t**2 - 0.234 * t + 3, 0)

# Create the Pyomo model
model = pyo.ConcreteModel()

# Sets
model.T = pyo.RangeSet(1, T)
model.N = pyo.RangeSet(0, N-1)
model.P = [(i, j) for i in range(N) for j in range(N) if (i, j) in pipe_capacities]

# Decision variables
model.x = pyo.Var(model.P, model.T, domain=pyo.NonNegativeReals)
model.y = pyo.Var(model.P, domain=pyo.Binary)
model.r = pyo.Var(model.N, model.T, domain=pyo.NonNegativeReals)

# Objective function
model.obj = pyo.Objective(expr=sum(model.x[4, 6, t] + model.x[5, 6, t] for t in model.T), sense=pyo.maximize)

# Constraints
def water_balance(model, i, t):
    if i == 0 or i == 6:
        return pyo.Constraint.Skip
    inflow = sum(model.x[j, i, t-1] for j in model.N if (j, i) in model.P)
    outflow = sum(model.x[i, j, t] for j in model.N if (i, j) in model.P)
    return inflow + model.r[i, t-1] == outflow + model.r[i, t]

model.water_balance = pyo.Constraint(model.N, model.T, rule=water_balance)

def water_inflow(model, t):
    outflow = sum(model.x[0, j, t] for j in model.N if (0, j) in model.P)
    return outflow + model.r[0, t] == f(t)

model.water_inflow = pyo.Constraint(model.T, rule=water_inflow)

def pipe_capacity(model, i, j, t):
    return model.x[i, j, t] <= pipe_capacities[i, j] * model.y[i, j]

model.pipe_capacity = pyo.Constraint(model.P, model.T, rule=pipe_capacity)

model.pipe_activation = pyo.Constraint(expr=sum(model.y[i, j] for (i, j) in model.P) <= K)

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

# Print the results
print("Objective value:", pyo.value(model.obj))
print("Activated pipes:")
for (i, j) in model.P:
    if pyo.value(model.y[i, j]) > 0.5:
        print(f"Pipe from {i} to {j}")
```

```
ERROR:pyomo.core:Rule failed when generating expression for Constraint water_balance with index (1, 1):
KeyError: "Index '(0, 1, 0)' is not valid for indexed component 'x'"
ERROR:pyomo.core:Constructing component 'water_balance' from data=None failed:
    KeyError: "Index '(0, 1, 0)' is not valid for indexed component 'x'"
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-11-924361c2bb79> in <cell line: 46>()
     44     return inflow + model.r[i, t-1] == outflow + model.r[i, t]
     45 
---> 46 model.water_balance = pyo.Constraint(model.N, model.T, rule=water_balance)
     47 
     48 def water_inflow(model, t):

8 frames
/usr/local/lib/python3.10/dist-packages/pyomo/core/base/indexed_component.py in _validate_index(self, idx)
    864         # Raise an exception
    865         #
--> 866         raise KeyError(
    867             "Index '%s' is not valid for indexed component '%s'"
    868             % (normalized_idx, self.name)

KeyError: "Index '(0, 1, 0)' is not valid for indexed component 'x'"
```

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

```
import pyomo.environ as pyo
import numpy as np
```

```
# Parameters
T = 10
N = 7
P = 12
K = 10

# Sample data for pipe capacities (c_ij)
# Replace these with the actual values from the problem
pipe_capacities = {
    (0, 1): 3, (0, 2): 6, (0, 5): 1, (1, 2): 3,
    (1, 3): 10, (1, 4): 4, (2, 4): 4, (2, 5): 4,
    (3, 4): 5, (4, 6): 5, (5, 1): 12, (5, 6): 7
}

# Water inflow function at point A
def f(t):
    return max(-4/30 * t**3 + t**2 - 0.234 * t + 3, 0)

# Create the Pyomo model
model = pyo.ConcreteModel()

# Sets
model.T = pyo.RangeSet(1, T)
model.N = pyo.RangeSet(0, N-1)
model.P = [(i, j) for i in range(N) for j in range(N) if (i, j) in pipe_capacities]

# Decision variables
model.x = pyo.Var(model.P, model.T, domain=pyo.NonNegativeReals)
model.y = pyo.Var(model.P, domain=pyo.Binary)
model.r = pyo.Var(model.N, model.T, domain=pyo.NonNegativeReals)

# Objective function
model.obj = pyo.Objective(expr=sum(model.x[4, 6, t] + model.x[5, 6, t] for t in model.T), sense=pyo.maximize)

# Constraints
def water_balance(model, i, t):
    if i == 0 or i == 6 or t == 1:
        return pyo.Constraint.Skip
    inflow = sum(model.x[j, i, t-1] for j in model.N if (j, i) in model.P)
    outflow = sum(model.x[i, j, t] for j in model.N if (i, j) in model.P)
    return inflow + model.r[i, t-1] == outflow + model.r[i, t]

model.water_balance = pyo.Constraint(model.N, model.T, rule=water_balance)

def water_inflow(model, t):
    outflow = sum(model.x[0, j, t] for j in model.N if (0, j) in model.P)
    return outflow + model.r[0, t] == f(t)

model.water_inflow = pyo.Constraint(model.T, rule=water_inflow)

def pipe_capacity(model, i, j, t):
    return model.x[i, j, t] <= pipe_capacities[i, j] * model.y[i, j]

model.pipe_capacity = pyo.Constraint(model.P, model.T, rule=pipe_capacity)

model.pipe_activation = pyo.Constraint(expr=sum(model.y[i, j] for (i, j) in model.P) <= K)

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

# Print the results
print("Objective value:", pyo.value(model.obj))
print("Activated pipes:")
for (i, j) in model.P:
    if pyo.value(model.y[i, j]) > 0.5:
        print(f"Pipe from {i} to {j}")
```

```
Objective value: 120.0
Activated pipes:
Pipe from 0 to 1
Pipe from 0 to 2
Pipe from 0 to 5
Pipe from 1 to 4
Pipe from 2 to 5
Pipe from 3 to 4
Pipe from 4 to 6
Pipe from 5 to 1
Pipe from 5 to 6
```

```
# Print the results
print("Objective value:", pyo.value(model.obj))
for i, j in model.P:
    if pyo.value(model.y[i, j]) > 0:
        print(f"Pipe ({i}, {j}) is activated")
        for t in model.T:
            print(f"Flow from {i} to {j} at time {t}: {pyo.value(model.x[i, j, t])}")
```

```
Objective value: 120.0
Pipe (0, 1) is activated
Flow from 0 to 1 at time 1: 0.0
Flow from 0 to 1 at time 2: 0.0
Flow from 0 to 1 at time 3: 3.0
Flow from 0 to 1 at time 4: 3.0
Flow from 0 to 1 at time 5: 3.0
Flow from 0 to 1 at time 6: 3.0
Flow from 0 to 1 at time 7: 0.0
Flow from 0 to 1 at time 8: 0.0
Flow from 0 to 1 at time 9: 0.0
Flow from 0 to 1 at time 10: 0.0
Pipe (0, 2) is activated
Flow from 0 to 2 at time 1: 0.0
Flow from 0 to 2 at time 2: 5.46533333333333
Flow from 0 to 2 at time 3: 3.698
Flow from 0 to 2 at time 4: 4.412
Flow from 0 to 2 at time 5: 6.0
Flow from 0 to 2 at time 6: 4.796
Flow from 0 to 2 at time 7: 3.62866666666667
Flow from 0 to 2 at time 8: 0.0
Flow from 0 to 2 at time 9: 0.0
Flow from 0 to 2 at time 10: 0.0
Pipe (0, 5) is activated
Flow from 0 to 5 at time 1: 0.0
Flow from 0 to 5 at time 2: 0.0
Flow from 0 to 5 at time 3: 1.0
Flow from 0 to 5 at time 4: 1.0
Flow from 0 to 5 at time 5: 1.0
Flow from 0 to 5 at time 6: 1.0
Flow from 0 to 5 at time 7: 1.0
Flow from 0 to 5 at time 8: 0.0
Flow from 0 to 5 at time 9: 0.0
Flow from 0 to 5 at time 10: 0.0
Pipe (1, 4) is activated
Flow from 1 to 4 at time 1: 0.0
Flow from 1 to 4 at time 2: 0.0
Flow from 1 to 4 at time 3: 4.0
Flow from 1 to 4 at time 4: 4.0
Flow from 1 to 4 at time 5: 4.0
Flow from 1 to 4 at time 6: 4.0
Flow from 1 to 4 at time 7: 4.0
Flow from 1 to 4 at time 8: 0.0
Flow from 1 to 4 at time 9: 0.0
Flow from 1 to 4 at time 10: 0.0
Pipe (2, 5) is activated
Flow from 2 to 5 at time 1: 0.0
Flow from 2 to 5 at time 2: 4.0
Flow from 2 to 5 at time 3: 4.0
Flow from 2 to 5 at time 4: 4.0
Flow from 2 to 5 at time 5: 4.0
Flow from 2 to 5 at time 6: 4.0
Flow from 2 to 5 at time 7: 4.0
Flow from 2 to 5 at time 8: 4.0
Flow from 2 to 5 at time 9: 4.0
Flow from 2 to 5 at time 10: 0.0
Pipe (3, 4) is activated
Flow from 3 to 4 at time 1: 0.0
```