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/MIP3.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 tasked with scheduling the power output of 6 electric power thermal units over the timespan of 15 periods. Ther
```

```
!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)
```

```
To formulate the mathematical optimization model for the given power plant scheduling problem, we need to define the par

Parameters:
- N: Number of power units (N = 6)
- T: Number of time periods (T = 15)
- SC_i: Startup cost for power unit i (€)
- SD_i: Shutdown cost for power unit i (€)
- FC_i: Fixed cost for power unit i (€)
- VC_i,t: Variable cost for power unit i at time t (€/MW)
- P_min_i: Minimum power output for unit i (MW)
- P_max_i: Maximum power output for unit i (MW)
- R_up_i: Maximum power increment for unit i (MW)
- R_down_i: Maximum power decrement for unit i (MW)
- D_t: Power demand at time t (MW)

Decision Variables:
- u_i,t: Binary variable indicating if unit i is on (1) or off (0) at time t
- p_i,t: Power output of unit i at time t (MW)
- v_i,t: Binary variable indicating if unit i is started up (1) or not (0) at time t
- w_i,t: Binary variable indicating if unit i is shut down (1) or not (0) at time t

Objective Function:
Minimize the total cost, which is the sum of startup costs, shutdown costs, fixed costs, and variable costs over all uni

min ∑_i=1^N ∑_t=1^T (SC_i * v_i,t + SD_i * w_i,t + FC_i * u_i,t + VC_i,t * p_i,t)

Constraints:
1. Power balance constraint: The total power output from all units should meet the demand plus 10% at each time period.
   ∑_i=1^N p_i,t = 1.1 * D_t, ∀t

2. Power output bounds: The power output of each unit should be within its minimum and maximum limits when the unit is o
   P_min_i * u_i,t ≤ p_i,t ≤ P_max_i * u_i,t, ∀i, ∀t

3. Ramp-up and ramp-down constraints: The change in power output between consecutive time periods should be within the m
   p_i,t - p_i,t-1 ≤ R_up_i * u_i,t-1 + P_max_i * v_i,t, ∀i, ∀t>1
   p_i,t-1 - p_i,t ≤ R_down_i * u_i,t + P_max_i * w_i,t, ∀i, ∀t>1

4. Startup and shutdown constraints: Ensure the correct values for the startup and shutdown variables based on the chang
   v_i,t - w_i,t = u_i,t - u_i,t-1, ∀i, ∀t>1
   v_i,t + w_i,t ≤ 1, ∀i, ∀t

5. Binary and non-negativity constraints:
   u_i,t, v_i,t, w_i,t ∈ {0, 1}, ∀i, ∀t
   p_i,t ≥ 0, ∀i, ∀t

This formulation represents the power plant scheduling problem as a mixed-integer linear programming (MILP) model, which
```

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.power_balance = Constraint(model.T, rule=power_balance_rule)

def power_output_min_rule(model, i, t):
    return model.p[i,t] >= P_min[i-1] * model.u[i,t]
model.power_output_min = Constraint(model.N, model.T, rule=power_output_min_rule)

def power_output_max_rule(model, i, t):
    return model.p[i,t] <= P_max[i-1] * model.u[i,t]
model.power_output_max = Constraint(model.N, model.T, rule=power_output_max_rule)

def ramp_up_rule(model, i, t):
    if t > 1:
        return model.p[i,t] - model.p[i,t-1] <= R_up[i-1] * model.u[i,t-1] + P_max[i-1] * model.v[i,t]
    else:
        return Constraint.Skip
model.ramp_up = Constraint(model.N, model.T, rule=ramp_up_rule)

def ramp_down_rule(model, i, t):
    if t > 1:
        return model.p[i,t-1] - model.p[i,t] <= R_down[i-1] * model.u[i,t] + P_max[i-1] * model.w[i,t]
    else:
        return Constraint.Skip
model.ramp_down = Constraint(model.N, model.T, rule=ramp_down_rule)

def startup_shutdown_rule(model, i, t):
    if t > 1:
        return model.v[i,t] - model.w[i,t] == model.u[i,t] - model.u[i,t-1]
    else:
        return Constraint.Skip
model.startup_shutdown = Constraint(model.N, model.T, rule=startup_shutdown_rule)

def startup_shutdown_limit_rule(model, i, t):
    return model.v[i,t] + model.w[i,t] <= 1
model.startup_shutdown_limit = Constraint(model.N, model.T, rule=startup_shutdown_limit_rule)

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

# Print the results
print("Objective value:", value(model.obj))

for t in model.T:
    print(f"Time period {t}:")
    for i in model.N:
        print(f"Unit {i}: Power output = {value(model.p[i,t]):.2f}, On/Off = {value(model.u[i,t])}")
    print()
```

In this code, I've used sample data for the parameters such as startup costs, shutdown costs, fixed costs, variable co

The code follows the mathematical formulation provided earlier. It defines the sets, decision variables, objective fun

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

```
from pyomo.environ import *

# Parameters
N = 6  # Number of power units
T = 15  # Number of time periods

# Sample data (replace with actual data)
SC = [10324, 5678, 7802, 12899, 4596, 9076]
SD = [2673, 5893, 982, 6783, 2596, 3561]
FC = [2000, 3000, 2500, 4000, 3500, 4500]
VC = [[20, 15, 18, 25, 22, 30],
      [22, 16, 19, 26, 23, 31],
      [23, 17, 20, 27, 24, 32],
      [24, 18, 21, 28, 25, 33],
      [25, 19, 22, 29, 26, 34],
      [26, 20, 23, 30, 27, 35],
      [27, 21, 24, 31, 28, 36],
      [28, 22, 25, 32, 29, 37],
      [29, 23, 26, 33, 30, 38],
      [30, 24, 27, 34, 31, 39],
```

```
       [31, 25, 28, 35, 32, 40],
       [32, 26, 29, 36, 33, 41],
       [33, 27, 30, 37, 34, 42],
       [34, 28, 31, 38, 35, 43],
       [35, 29, 32, 39, 36, 44],
       ]
P_min = [50, 40, 30, 60, 55, 65]
P_max = [500, 600, 550, 700, 650, 750]
R_up = [100, 120, 110, 130, 125, 140]
R_down = [90, 110, 100, 120, 115, 130]
D = [1000, 1200, 1300, 1100, 1500, 1400, 1600, 1300, 1700, 1800, 1900, 1600, 2000, 1800, 1700]

# Create the Pyomo model
model = ConcreteModel()

# Define sets
model.N = RangeSet(1, N)
model.T = RangeSet(1, T)

# Define decision variables
model.u = Var(model.N, model.T, domain=Binary)
model.p = Var(model.N, model.T, domain=NonNegativeReals)
model.v = Var(model.N, model.T, domain=Binary)
model.w = Var(model.N, model.T, domain=Binary)

# Define objective function
def obj_rule(model):
    return sum(SC[i-1] * model.v[i,t] + SD[i-1] * model.w[i,t] + FC[i-1] * model.u[i,t] + VC[t-1][i-1] * model.p[i,t] for i i
model.obj = Objective(rule=obj_rule, sense=minimize)

# Define constraints
def power_balance_rule(model, t):
    return sum(model.p[i,t] for i in model.N) == 1.1 * D[t-1]
model.power_balance = Constraint(model.T, rule=power_balance_rule)

def power_output_min_rule(model, i, t):
    return model.p[i,t] >= P_min[i-1] * model.u[i,t]
model.power_output_min = Constraint(model.N, model.T, rule=power_output_min_rule)

def power_output_max_rule(model, i, t):
    return model.p[i,t] <= P_max[i-1] * model.u[i,t]
model.power_output_max = Constraint(model.N, model.T, rule=power_output_max_rule)

def ramp_up_rule(model, i, t):
    if t > 1:
        return model.p[i,t] - model.p[i,t-1] <= R_up[i-1] * model.u[i,t-1] + P_max[i-1] * model.v[i,t]
    else:
        return Constraint.Skip
model.ramp_up = Constraint(model.N, model.T, rule=ramp_up_rule)

def ramp_down_rule(model, i, t):
    if t > 1:
        return model.p[i,t-1] - model.p[i,t] <= R_down[i-1] * model.u[i,t] + P_max[i-1] * model.w[i,t]
    else:
        return Constraint.Skip
model.ramp_down = Constraint(model.N, model.T, rule=ramp_down_rule)

def startup_shutdown_rule(model, i, t):
    if t > 1:
        return model.v[i,t] - model.w[i,t] == model.u[i,t] - model.u[i,t-1]
    else:
        return Constraint.Skip
model.startup_shutdown = Constraint(model.N, model.T, rule=startup_shutdown_rule)

def startup_shutdown_limit_rule(model, i, t):
    return model.v[i,t] + model.w[i,t] <= 1
model.startup_shutdown_limit = Constraint(model.N, model.T, rule=startup_shutdown_limit_rule)

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

# Print the results
print("Objective value:", value(model.obj))

for t in model.T:
    print(f"Time period {t}:")
    for i in model.N:
        print(f"Unit {i}: Power output = {value(model.p[i,t]):.2f}, On/Off = {value(model.u[i,t])}")
    print()
```

```
Objective value: 806126.0
Time period 1:
Unit 1: Power output = 80.00, On/Off = 1.0
Unit 2: Power output = 590.00, On/Off = 1.0
Unit 3: Power output = 430.00, On/Off = 1.0
Unit 4: Power output = 0.00, On/Off = 0.0
Unit 5: Power output = 0.00, On/Off = 0.0
Unit 6: Power output = 0.00, On/Off = 0.0

Time period 2:
Unit 1: Power output = 180.00, On/Off = 1.0
Unit 2: Power output = 600.00, On/Off = 1.0
Unit 3: Power output = 540.00, On/Off = 1.0
Unit 4: Power output = 0.00, On/Off = 0.0
Unit 5: Power output = 0.00, On/Off = 0.0
Unit 6: Power output = 0.00, On/Off = 0.0

Time period 3:
Unit 1: Power output = 280.00, On/Off = 1.0
Unit 2: Power output = 600.00, On/Off = 1.0
Unit 3: Power output = 550.00, On/Off = 1.0
Unit 4: Power output = 0.00, On/Off = 0.0
Unit 5: Power output = 0.00, On/Off = 0.0
Unit 6: Power output = 0.00, On/Off = 0.0

Time period 4:
Unit 1: Power output = 190.00, On/Off = 1.0
Unit 2: Power output = 570.00, On/Off = 1.0
Unit 3: Power output = 450.00, On/Off = 1.0
Unit 4: Power output = 0.00, On/Off = 0.0
Unit 5: Power output = 0.00, On/Off = 0.0
Unit 6: Power output = 0.00, On/Off = 0.0

Time period 5:
Unit 1: Power output = 290.00, On/Off = 1.0
Unit 2: Power output = 600.00, On/Off = 1.0
Unit 3: Power output = 550.00, On/Off = 1.0
Unit 4: Power output = 0.00, On/Off = 0.0
Unit 5: Power output = 210.00, On/Off = 1.0
Unit 6: Power output = 0.00, On/Off = 0.0

Time period 6:
Unit 1: Power output = 295.00, On/Off = 1.0
Unit 2: Power output = 600.00, On/Off = 1.0
Unit 3: Power output = 550.00, On/Off = 1.0
Unit 4: Power output = 0.00, On/Off = 0.0
Unit 5: Power output = 95.00, On/Off = 1.0
Unit 6: Power output = 0.00, On/Off = 0.0

Time period 7:
Unit 1: Power output = 395.00, On/Off = 1.0
Unit 2: Power output = 600.00, On/Off = 1.0
Unit 3: Power output = 545.00, On/Off = 1.0
Unit 4: Power output = 0.00, On/Off = 0.0
Unit 5: Power output = 220.00, On/Off = 1.0
Unit 6: Power output = 0.00, On/Off = 0.0

Time period 8:
```

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