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)
```

```
Here's the mathematical optimization model for the given problem:

Indices:
i: power plant units (i = 1, 2, ..., 6)
t: time periods (t = 1, 2, ..., 15)

Parameters:
SC_i: startup cost for power plant i
SD_i: shutdown cost for power plant i
FC_i: fixed cost for power plant i when running
VC_i,t: variable cost for power plant i at time t
P_min_i: minimum power output for power plant i
P_max_i: maximum power output for power plant i
R_up_i: maximum power increment for power plant i
R_down_i: maximum power decrement for power plant i
D_t: power demand at time t

Decision Variables:
x_i,t: power output of unit i at time t (continuous)
u_i,t: binary variable indicating if unit i is running at time t
v_i,t: binary variable indicating if unit i is started up at time t
w_i,t: binary variable indicating if unit i is shut down at time t

Objective Function:
minimize ∑_i ∑_t (FC_i * u_i,t + VC_i,t * x_i,t + SC_i * v_i,t + SD_i * w_i,t)

Constraints:
1. Power balance constraint:
   ∑_i x_i,t = 1.1 * D_t, ∀t

2. Power output limits:
   P_min_i * u_i,t ≤ x_i,t ≤ P_max_i * u_i,t, ∀i, ∀t

3. Ramp up and ramp down constraints:
   x_i,t - x_i,t-1 ≤ R_up_i * u_i,t-1 + P_max_i * v_i,t, ∀i, ∀t > 1
   x_i,t-1 - x_i,t ≤ R_down_i * u_i,t + P_max_i * w_i,t, ∀i, ∀t > 1

4. Logical constraints for startup and shutdown:
   u_i,t - u_i,t-1 = v_i,t - w_i,t, ∀i, ∀t > 1
   v_i,t + w_i,t ≤ 1, ∀i, ∀t

5. Initial conditions:
   u_i,1 - u_i,0 = v_i,1 - w_i,1, ∀i
   u_i,0 given

6. Binary constraints:
   u_i,t, v_i,t, w_i,t ∈ {0, 1}, ∀i, ∀t

The objective function minimizes the total cost, which includes fixed costs, variable costs, startup costs, and shutdo

The constraints ensure that:
1. The total power output meets the demand plus a 10% security margin.
2. The power output of each unit is within its minimum and maximum limits.
3. The power output changes between consecutive time periods are within the ramp up and ramp down limits.
4. The startup and shutdown variables are logically linked to the unit commitment variables.
5. The initial conditions are met.
6. The unit commitment, startup, and shutdown variables are binary.
```

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.objective = pyo.Objective(rule=objective_rule, sense=pyo.minimize)

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

def power_output_limits_rule(model, i, t):
    return (P_min[i-1] * model.u[i, t], model.x[i, t], P_max[i-1] * model.u[i, t])
model.power_output_limits = pyo.Constraint(model.units, model.periods, rule=power_output_limits_rule)

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

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

def logical_constraints_rule(model, i, t):
    if t > 1:
        return model.u[i, t] - model.u[i, t-1] == model.v[i, t] - model.w[i, t]
    else:
        return pyo.Constraint.Skip
model.logical_constraints = pyo.Constraint(model.units, model.periods, rule=logical_constraints_rule)

def startup_shutdown_rule(model, i, t):
    return model.v[i, t] + model.w[i, t] <= 1
model.startup_shutdown = pyo.Constraint(model.units, model.periods, rule=startup_shutdown_rule)

def initial_conditions_rule(model, i):
    return model.u[i, 1] - initial_state[i-1] == model.v[i, 1] - model.w[i, 1]
model.initial_conditions = pyo.Constraint(model.units, rule=initial_conditions_rule)

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

# Print the results
print("Objective value:", pyo.value(model.objective))
for t in model.periods:
    print(f"Time period {t}:")
    for i in model.units:
        print(f"Unit {i}: Power output = {pyo.value(model.x[i, t])}, Status = {pyo.value(model.u[i, t])}")
```

Note: In this code, I used sample data for the parameter values. Replace them with the actual data for your specific p
- `SC`, `SD`, `FC`: startup costs, shutdown costs, and fixed costs for each power plant
- `VC`: variable costs for each power plant at each time period
- `P_min`, `P_max`: minimum and maximum power output for each power plant
- `R_up`, `R_down`: maximum power increment and decrement for each power plant
- `D`: power demand at each time period

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
```

```
# Sample data (replace with actual data)
num_units = 6
num_periods = 15

# 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]
initial_state = [1, 1, 0, 0, 1, 0]

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

# Define sets
model.units = pyo.RangeSet(1, num_units)
model.periods = pyo.RangeSet(1, num_periods)

# Define decision variables
model.x = pyo.Var(model.units, model.periods, domain=pyo.NonNegativeReals)
model.u = pyo.Var(model.units, model.periods, domain=pyo.Binary)
model.v = pyo.Var(model.units, model.periods, domain=pyo.Binary)
model.w = pyo.Var(model.units, model.periods, domain=pyo.Binary)

# Define objective function
def objective_rule(model):
    return sum(FC[i-1] * model.u[i, t] + VC[t-1][i-1] * model.x[i, t] + SC[i-1] * model.v[i, t] + SD[i-1] * model.w[i, t]
               for i in model.units for t in model.periods)
model.objective = pyo.Objective(rule=objective_rule, sense=pyo.minimize)

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

def power_output_limits_rule(model, i, t):
    return (P_min[i-1] * model.u[i, t], model.x[i, t], P_max[i-1] * model.u[i, t])
model.power_output_limits = pyo.Constraint(model.units, model.periods, rule=power_output_limits_rule)

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

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

def logical_constraints_rule(model, i, t):
    if t > 1:
        return model.u[i, t] - model.u[i, t-1] == model.v[i, t] - model.w[i, t]
    else:
        return pyo.Constraint.Skip
model.logical_constraints = pyo.Constraint(model.units, model.periods, rule=logical_constraints_rule)

def startup_shutdown_rule(model, i, t):
    return model.v[i, t] + model.w[i, t] <= 1
```

```
    return model.v[i, t] + model.w[i, t] <= 1
model.startup_shutdown = pyo.Constraint(model.units, model.periods, rule=startup_shutdown_rule)

def initial_conditions_rule(model, i):
    return model.u[i, 1] - initial_state[i-1] == model.v[i, 1] - model.w[i, 1]
model.initial_conditions = pyo.Constraint(model.units, rule=initial_conditions_rule)

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

# Print the results
print("Objective value:", pyo.value(model.objective))
for t in model.periods:
    print(f"Time period {t}:")
    for i in model.units:
        print(f"Unit {i}: Power output = {pyo.value(model.x[i, t])}, Status = {pyo.value(model.u[i, t])}")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-efdff36c4994> in <cell line: 95>()
     93 # Solve the optimization problem
     94 solver = pyo.SolverFactory('glpk')
---> 95 results = solver.solve(model)
     96
     97 # Print the results

                                11 frames
/usr/local/lib/python3.10/dist-packages/pyomo/core/base/constraint.py in
_get_range_bound(self, range_arg)
    205         bound = self._expr.arg(range_arg)
    206         if not is_fixed(bound):
--> 207             raise ValueError(
    208                 "Constraint '%s' is a Ranged Inequality with a "
    209                 "variable %s bound.  Cannot normalize the "

ValueError: Constraint 'power_output_limits[1,1]' is a Ranged Inequality with
a variable lower bound.  Cannot normalize the constraint or send it to a
```

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

```
import pyomo.environ as pyo

# Sample data (replace with actual data)
num_units = 6
num_periods = 15

# Sample data (replace with actual data)
N = 6
T = 15
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]
initial_state = [1, 1, 0, 0, 1, 0]

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

# Define sets
model.units = pyo.RangeSet(1, num_units)
model.periods = pyo.RangeSet(1, num_periods)
```

```
# Define decision variables
model.x = pyo.Var(model.units, model.periods, domain=pyo.NonNegativeReals)
model.u = pyo.Var(model.units, model.periods, domain=pyo.Binary)
model.v = pyo.Var(model.units, model.periods, domain=pyo.Binary)
model.w = pyo.Var(model.units, model.periods, domain=pyo.Binary)

# Define objective function
def objective_rule(model):
    return sum(FC[i-1] * model.u[i, t] + VC[t-1][i-1] * model.x[i, t] + SC[i-1] * model.v[i, t] + SD[i-1] * model.w[i, t]
               for i in model.units for t in model.periods)
model.objective = pyo.Objective(rule=objective_rule, sense=pyo.minimize)

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

def power_output_limits_rule_lower(model, i, t): # MODIFIED THE CONSTRAINT FOR PROPER SYNTAX
    return P_min[i-1] * model.u[i, t] <= model.x[i, t]
model.power_output_limits_lower = pyo.Constraint(model.units, model.periods, rule=power_output_limits_rule_lower)

def power_output_limits_rule_upper(model, i, t): # MODIFIED THE CONSTRAINT FOR PROPER SYNTAX
    return model.x[i, t] <= P_max[i-1] * model.u[i, t]
model.power_output_limits_upper = pyo.Constraint(model.units, model.periods, rule=power_output_limits_rule_upper)

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

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

def logical_constraints_rule(model, i, t):
    if t > 1:
        return model.u[i, t] - model.u[i, t-1] == model.v[i, t] - model.w[i, t]
    else:
        return pyo.Constraint.Skip
model.logical_constraints = pyo.Constraint(model.units, model.periods, rule=logical_constraints_rule)

def startup_shutdown_rule(model, i, t):
    return model.v[i, t] + model.w[i, t] <= 1
model.startup_shutdown = pyo.Constraint(model.units, model.periods, rule=startup_shutdown_rule)

def initial_conditions_rule(model, i):
    return model.u[i, 1] - initial_state[i-1] == model.v[i, 1] - model.w[i, 1]
model.initial_conditions = pyo.Constraint(model.units, rule=initial_conditions_rule)

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

# Print the results
print("Objective value:", pyo.value(model.objective))
for t in model.periods:
    print(f"Time period {t}:")
    for i in model.units:
        print(f"Unit {i}: Power output = {pyo.value(model.x[i, t])}, Status = {pyo.value(model.u[i, t])}")
```

```
Unit 4: Power output = 0.0, Status = 0.0
Unit 5: Power output = 220.0, Status = 1.0
Unit 6: Power output = 0.0, Status = 0.0
Time period 10:
Unit 1: Power output = 500.0, Status = 1.0
Unit 2: Power output = 600.0, Status = 1.0
Unit 3: Power output = 550.0, Status = 1.0
Unit 4: Power output = 0.0, Status = 0.0
Unit 5: Power output = 330.0, Status = 1.0
Unit 6: Power output = 0.0, Status = 0.0
Time period 11:
Unit 1: Power output = 490.0, Status = 1.0
Unit 2: Power output = 600.0, Status = 1.0
Unit 3: Power output = 545.0, Status = 1.0
Unit 4: Power output = 0.0, Status = 0.0
Unit 5: Power output = 455.0, Status = 1.0
Unit 6: Power output = 0.0, Status = 0.0
Time period 12:
Unit 1: Power output = 400.0, Status = 1.0
Unit 2: Power output = 490.0, Status = 1.0
Unit 3: Power output = 445.0, Status = 1.0
Unit 4: Power output = 0.0, Status = 0.0
Unit 5: Power output = 425.0, Status = 1.0
Unit 6: Power output = 0.0, Status = 0.0
Time period 13:
Unit 1: Power output = 500.0, Status = 1.0
Unit 2: Power output = 600.0, Status = 1.0
Unit 3: Power output = 550.0, Status = 1.0
Unit 4: Power output = 0.0, Status = 0.0
Unit 5: Power output = 550.0, Status = 1.0
Unit 6: Power output = 0.0, Status = 0.0
Time period 14:
Unit 1: Power output = 410.0, Status = 1.0
Unit 2: Power output = 600.0, Status = 1.0
Unit 3: Power output = 535.0, Status = 1.0
Unit 4: Power output = 0.0, Status = 0.0
Unit 5: Power output = 435.0, Status = 1.0
Unit 6: Power output = 0.0, Status = 0.0
Time period 15:
Unit 1: Power output = 400.0, Status = 1.0
```