## 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
# Variables Prompt
prompt11_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt11_MathematicalModel.txt'

# Objective Prompt
prompt12_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt12_MathematicalModel.txt'

# Constraint Prompt
prompt13_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt13_MathematicalModel.txt'

# Code Prompt
prompt2_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt2_PyomoCode.txt'
problem_desc_path = '/content/drive/MyDrive/Thesis/ProblemDescriptions/MIP/MIP3.txt'

prompt11_file = open(prompt11_path, "r")
prompt12_file = open(prompt12_path, "r")
prompt13_file = open(prompt13_path, "r")
prompt2_file = open(prompt2_path, "r")
problem_desc_file = open(problem_desc_path, "r")

prompt11 = prompt11_file.read()
print("Prompt 1.1 (Variables):\n", prompt11)

prompt12 = prompt12_file.read()
print("Prompt 1.2 (Objctive):\n", prompt12)

prompt13 = prompt13_file.read()
print("Prompt 1.3 (Constraints):\n", prompt13)

prompt2 = prompt2_file.read()
print("Prompt 2:\n", prompt2)

problem_desc = problem_desc_file.read()
print("Problem Description:\n", problem_desc)
```

```
Prompt 1.1 (Variables):
 Please formulate only the variables for this mathematical optimization problem.
Prompt 1.2 (Objctive):
 Please formulate only the objective function for this mathematical optimization problem.
Prompt 1.3 (Constraints):
 Please formulate only the constraints for this mathematical optimization problem.
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 1.1: Create Variables for Mathematical Model

```
message11 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt11,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)

response_p11 = message11.content[0].text


# Print response
print(response_p11)
```

```
Here are the variables for this mathematical optimization problem:

Let i = 1, ..., 6 denote the power plant units
Let t = 1, ..., 15 denote the time periods

Decision variables:
p(i,t) = power output of unit i at time t
u(i,t) = binary variable indicating if unit i is running (1) or not (0) at time t
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

Parameters:
SC(i) = startup cost for unit i
SD(i) = shutdown cost for unit i
FC(i) = fixed cost for operating unit i
VC(i,t) = variable cost coefficient for unit i at time t
P_min(i) = minimum power output of unit i when running
P_max(i) = maximum power output of unit i when running
R_up(i) = maximum power increment for unit i between periods
R_down(i) = maximum power decrement for unit i between periods
D(t) = total power demand at time t

The variables u(i,t), v(i,t) and w(i,t) capture the on/off status and startups/shutdowns of the units over time, while p
```

## 1. Prompt 1.2: Create Objective for Mathematical Model

```
message12 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt12,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11)}
    ]
)

response_p12 = message12.content[0].text


# Print response
print(response_p12)
```

```
Based on the provided problem description and notation, the objective function for this unit commitment problem can be f

minimize sum(i=1 to 6) sum(t=1 to 15) [FC(i)*u(i,t) + VC(i,t)*p(i,t) + SC(i)*v(i,t) + SD(i)*w(i,t)]

This objective function aims to minimize the total cost of operating the 6 power units over the 15 time periods, which c
- Fixed costs FC(i) applied when a unit i is running (u(i,t)=1)
- Variable costs VC(i,t) proportional to the power output p(i,t) of unit i
- Startup costs SC(i) incurred when a unit i is started up (v(i,t)=1)
- Shutdown costs SD(i) incurred when a unit i is shut down (w(i,t)=1)

The sum over i=1 to 6 aggregates the costs across all power units, while the sum over t=1 to 15 aggregates the costs acr

This objective is subject to various constraints on the p(i,t), u(i,t), v(i,t) and w(i,t) variables, including:
```

```
- Minimum and maximum output limits when a unit is running
- Ramping up/down limits between consecutive periods
- Logic relating the u, v, w variables for consistent on/off/startup/shutdown states
- Demand satisfaction and 10% excess capacity requirements in each period
But the objective function itself covers the cost minimization goal as described above. Let me know if you need any clar
```

## 1. Prompt 1.3: Create Constraints for Mathematical Model

```
message13 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt13,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11 + response_p12)}
    ]
)

response_p13 = message13.content[0].text


# Print response
print(response_p13)
```

```
Thank you for providing the problem description and the objective function. Here are the constraints for this unit commi

1. Power output bounds:
   P_min(i) * u(i,t) <= p(i,t) <= P_max(i) * u(i,t), for all i, t
   These constraints ensure that the power output of each unit i at time t is within its minimum and maximum limits when

2. Ramping constraints:
   p(i,t) - p(i,t-1) <= R_up(i), for all i, t > 1
   p(i,t-1) - p(i,t) <= R_down(i), for all i, t > 1
   These constraints limit the change in power output of each unit i between consecutive time periods based on the maxim

3. Logical status constraints:
   u(i,t) - u(i,t-1) = v(i,t) - w(i,t), for all i, t > 1
   v(i,t) + w(i,t) <= 1, for all i, t
   These constraints establish the logical relationships between the u, v, and w variables. The first constraint ensures

4. Demand satisfaction constraint:
   sum(i=1 to 6) p(i,t) = D(t), for all t
   This constraint ensures that the total power output from all units meets the demand D(t) in each time period t.

5. Excess capacity constraint:
   sum(i=1 to 6) P_max(i) * u(i,t) >= 1.1 * D(t), for all t
   This constraint ensures that the total available power capacity (considering the maximum output of running units) is

6. Binary variable constraints:
   u(i,t), v(i,t), w(i,t) are binary, for all i, t

These constraints, along with the objective function you provided, complete the formulation of the unit commitment probl
```

## 2. Prompt 2: Write Code based on Mathematical Model

In case more than the component of each step was generated, use "last instance"/ "newest version" of the components.

Example: Prompt 1.1 (Variables): Model generates only variables Prompt 1.2 (Objectvie): Model generates objective and constraints Prompt 1.3 (Constraints): Model generates only constraints

Use Variables from step 1.1, use objective from 1.2 and use constraints from 1.3 (since constraints version 1.3 is more recent than constraints version 1.2).

This set up allows for potential self-correction of the model throughout the three prompts.

Correctness of each component is still generated in the respective step, so self-correction is independent of this.

```
message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": (response_p11 + response_p12 + response_p13)}
    ]
)

response_p2 = message2.content[0].text
```

```
# Print response
print(response_p2)
```

Here's a Python code using Pyomo for the unit commitment optimization problem:

```python
import pyomo.environ as pyo

# Define the model
model = pyo.ConcreteModel()

# Define sets
model.I = pyo.RangeSet(1, 6)  # Set of power plant units
model.T = pyo.RangeSet(1, 15)  # Set of time periods

# Define parameters
model.SC = pyo.Param(model.I, initialize={1: 100, 2: 150, 3: 200, 4: 120, 5: 180, 6: 250})  # Startup costs (sample da
model.SD = pyo.Param(model.I, initialize={1: 50, 2: 80, 3: 100, 4: 60, 5: 90, 6: 120})  # Shutdown costs (sample data)
model.FC = pyo.Param(model.I, initialize={1: 500, 2: 600, 3: 700, 4: 550, 5: 650, 6: 750})  # Fixed costs (sample data
model.VC = pyo.Param(model.I, model.T, initialize={(i, t): 10 + 0.5 * i + 0.1 * t for i in model.I for t in model.T})
model.P_min = pyo.Param(model.I, initialize={1: 50, 2: 60, 3: 70, 4: 55, 5: 65, 6: 75})  # Minimum power output (sampl
model.P_max = pyo.Param(model.I, initialize={1: 200, 2: 250, 3: 300, 4: 225, 5: 275, 6: 325})  # Maximum power output
model.R_up = pyo.Param(model.I, initialize={1: 50, 2: 60, 3: 70, 4: 55, 5: 65, 6: 75})  # Maximum power increment (sam
model.R_down = pyo.Param(model.I, initialize={1: 50, 2: 60, 3: 70, 4: 55, 5: 65, 6: 75})  # Maximum power decrement (s
model.D = pyo.Param(model.T, initialize={t: 500 + 50 * t for t in model.T})  # Power demand (sample data)

# Define variables
model.p = pyo.Var(model.I, model.T, within=pyo.NonNegativeReals)  # Power output
model.u = pyo.Var(model.I, model.T, within=pyo.Binary)  # Unit status
model.v = pyo.Var(model.I, model.T, within=pyo.Binary)  # Startup status
model.w = pyo.Var(model.I, model.T, within=pyo.Binary)  # Shutdown status

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

# Define constraints
def power_output_bounds(model, i, t):
    return model.P_min[i] * model.u[i, t] <= model.p[i, t] <= model.P_max[i] * model.u[i, t]
model.power_output_bounds = pyo.Constraint(model.I, model.T, rule=power_output_bounds)

def ramping_up(model, i, t):
    if t > 1:
        return model.p[i, t] - model.p[i, t-1] <= model.R_up[i]
    else:
        return pyo.Constraint.Skip
model.ramping_up = pyo.Constraint(model.I, model.T, rule=ramping_up)

def ramping_down(model, i, t):
    if t > 1:
        return model.p[i, t-1] - model.p[i, t] <= model.R_down[i]
    else:
        return pyo.Constraint.Skip
model.ramping_down = pyo.Constraint(model.I, model.T, rule=ramping_down)

def logical_status(model, i, t):
    if t > 1:
```

## 4. Input Problem Data and Test Model Code

```
# Installing pyomo nd solver
!pip install -q pyomo
!pip install pandas
!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 sets
model.I = pyo.RangeSet(1, 6)  # Set of power plant units
model.T = pyo.RangeSet(1, 15)  # Set of time periods

# Define parameters
model.SC = pyo.Param(model.I, initialize={1: 10324, 2: 5678, 3: 7802, 4: 12899, 5: 4596, 6: 9076})  # Startup costs (sample
model.SD = pyo.Param(model.I, initialize={1: 2673, 2: 5893, 3: 982, 4: 6783, 5: 2596, 6: 3561})  # Shutdown costs (sample da
model.FC = pyo.Param(model.I, initialize={1: 2000, 2: 3000, 3: 2500, 4: 4000, 5: 3500, 6: 4500})  # Fixed costs (sample data
model.VC = pyo.Param(model.I, model.T, initialize={(1,1): 20, (1,2): 22, (1,3): 23, (1,4): 24, (1,5): 25, (1,6): 26, (1,7):
      (2,1): 15, (2,2): 16, (2,3): 17, (2,4): 18, (2,5): 19, (2,6): 20, (2,7): 21, (2,8): 22, (2,9): 23, (2,10): 24, (2,11):
      (3,1): 18, (3,2): 19, (3,3): 20, (3,4): 21, (3,5): 22, (3,6): 23, (3,7): 24, (3,8): 25, (3,9): 26, (3,10): 27, (3,11):
      (4,1): 25, (4,2): 26, (4,3): 27, (4,4): 28, (4,5): 29, (4,6): 30, (4,7): 31, (4,8): 32, (4,9): 33, (4,10): 34, (4,11):
      (5,1): 22, (5,2): 23, (5,3): 24, (5,4): 25, (5,5): 26, (5,6): 27, (5,7): 28, (5,8): 29, (5,9): 30, (5,10): 31, (5,11):
      (6,1): 30, (6,2): 31, (6,3): 32, (6,4): 33, (6,5): 34, (6,6): 35, (6,7): 36, (6,8): 37, (6,9): 38, (6,10): 39, (6,11):
})  # Variable costs (sample data)
model.P_min = pyo.Param(model.I, initialize={1: 50, 2: 40, 3: 30, 4: 60, 5: 55, 6: 65})  # Minimum power output (sample data
model.P_max = pyo.Param(model.I, initialize={1: 500, 2: 600, 3: 550, 4: 700, 5: 650, 6: 750})  # Maximum power output (sampl
model.R_up = pyo.Param(model.I, initialize={1: 100, 2: 120, 3: 110, 4: 130, 5: 125, 6: 140})  # Maximum power increment (sam
model.R_down = pyo.Param(model.I, initialize={1: 90, 2: 110, 3: 100, 4: 120, 5: 115, 6: 130})  # Maximum power decrement (sa
model.D = pyo.Param(model.T, initialize={1: 280, 2: 327, 3: 400, 4: 388, 5: 501, 6: 600, 7: 800, 8: 927, 9: 705, 10: 502, 11

# Define variables
model.p = pyo.Var(model.I, model.T, within=pyo.NonNegativeReals)  # Power output
model.u = pyo.Var(model.I, model.T, within=pyo.Binary)  # Unit status
model.v = pyo.Var(model.I, model.T, within=pyo.Binary)  # Startup status
model.w = pyo.Var(model.I, model.T, within=pyo.Binary)  # Shutdown status

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

# Define constraints
def power_output_bounds(model, i, t):
    return model.P_min[i] * model.u[i, t] <= model.p[i, t] <= model.P_max[i] * model.u[i, t]
model.power_output_bounds = pyo.Constraint(model.I, model.T, rule=power_output_bounds)

def ramping_up(model, i, t):
    if t > 1:
        return model.p[i, t] - model.p[i, t-1] <= model.R_up[i]
    else:
        return pyo.Constraint.Skip
model.ramping_up = pyo.Constraint(model.I, model.T, rule=ramping_up)

def ramping_down(model, i, t):
    if t > 1:
        return model.p[i, t-1] - model.p[i, t] <= model.R_down[i]
    else:
        return pyo.Constraint.Skip
model.ramping_down = pyo.Constraint(model.I, model.T, rule=ramping_down)

def logical_status(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_status = pyo.Constraint(model.I, model.T, rule=logical_status)

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

def demand_satisfaction(model, t):
    return sum(model.p[i, t] for i in model.I) == model.D[t]
model.demand_satisfaction = pyo.Constraint(model.T, rule=demand_satisfaction)

def excess_capacity(model, t):
    return sum(model.P_max[i] * model.u[i, t] for i in model.I) >= 1.1 * model.D[t]
model.excess_capacity = pyo.Constraint(model.T, rule=excess_capacity)

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

# Print the results
```

```
print("Optimal Objective Value:", pyo.value(model.obj))
for i in model.I:
    for t in model.T:
        print(f"Unit {i} at time {t}: Power = {model.p[i, t].value}, Status = {model.u[i, t].value}")
```

```
ERROR:pyomo.core:Rule failed when generating expression for Constraint power_(
PyomoException: Cannot convert non-constant Pyomo expression (50*u[1,1]  <=  |
This error is usually caused by using a Var, unit, or mutable Param in a
Boolean context such as an "if" statement, or when checking container
membership or equality. For example,
    >>> m.x = Var()
    >>> if m.x >= 1:
    ...     pass
and
    >>> m.y = Var()
    >>> if m.y in [m.x, m.y]:
    ...     pass
would both cause this exception.
ERROR:pyomo.core:Constructing component 'power_output_bounds' from data=None
    PyomoException: Cannot convert non-constant Pyomo expression (50*u[1,1]  <
This error is usually caused by using a Var, unit, or mutable Param in a
Boolean context such as an "if" statement, or when checking container
membership or equality. For example,
    >>> m.x = Var()
    >>> if m.x >= 1:
    ...     pass
and
    >>> m.y = Var()
    >>> if m.y in [m.x, m.y]:
    ...     pass
would both cause this exception.
---------------------------------------------------------------------------
PyomoException                            Traceback (most recent call last)
<ipython-input-16-2591f2473f9b> in <cell line: 43>()
     41 def power_output_bounds(model, i, t):
     42     return model.P_min[i] * model.u[i, t] <= model.p[i, t] <=
model.P_max[i] * model.u[i, t]
---> 43 model.power_output_bounds = pyo.Constraint(model.I, model.T,
rule=power_output_bounds)
     44
     45 def ramping_up(model, i, t):

                                5 frames
/usr/local/lib/python3.10/dist-packages/pyomo/core/expr/relational_expr.py in
__bool__(self)
     45         if self.is_constant():
     46             return bool(self())
---> 47         raise PyomoException(
     48             """
     49 Cannot convert non-constant Pyomo expression (%s) to bool.

PyomoException: Cannot convert non-constant Pyomo expression (50*u[1,1]  <=
p[1,1]) to bool.
This error is usually caused by using a Var, unit, or mutable Param in a
Boolean context such as an "if" statement, or when checking container
membership or equality. For example,
    >>> m.x = Var()
    >>> if m.x >= 1:
    ...     pass
and
    >>> m.y = Var()
```

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

```
# Download Gurobi
!wget https://packages.gurobi.com/9.5/gurobi9.5.2_linux64.tar.gz

# Extract the tarball
!tar -xvzf gurobi9.5.2_linux64.tar.gz

# Set up environment variables for Gurobi
import os
os.environ['GUROBI_HOME'] = "/content/gurobi952/linux64"
os.environ['PATH'] += ":/content/gurobi952/linux64/bin"
os.environ['LD_LIBRARY_PATH'] = "/content/gurobi952/linux64/lib"
```

Show hidden output

```
import shutil
shutil.move('/content/drive/MyDrive/gurobi.lic', '/root/gurobi.lic')
```

```
'/root/gurobi.lic'
```

```
import pyomo.environ as pyo

# Define the model
model = pyo.ConcreteModel()

# Define sets
model.I = pyo.RangeSet(1, 6)  # Set of power plant units
model.T = pyo.RangeSet(1, 15)  # Set of time periods

# Define parameters
model.SC = pyo.Param(model.I, initialize={1: 10324, 2: 5678, 3: 7802, 4: 12899, 5: 4596, 6: 9076})  # Startup costs (sample d
model.SD = pyo.Param(model.I, initialize={1: 2673, 2: 5893, 3: 982, 4: 6783, 5: 2596, 6: 3561})  # Shutdown costs (sample dat
model.FC = pyo.Param(model.I, initialize={1: 2000, 2: 3000, 3: 2500, 4: 4000, 5: 3500, 6: 4500})  # Fixed costs (sample data)
model.VC = pyo.Param(model.I, model.T, initialize={(1,1): 20, (1,2): 22, (1,3): 23, (1,4): 24, (1,5): 25, (1,6): 26, (1,7): 2
      (2,1): 15, (2,2): 16, (2,3): 17, (2,4): 18, (2,5): 19, (2,6): 20, (2,7): 21, (2,8): 22, (2,9): 23, (2,10): 24, (2,11):
      (3,1): 18, (3,2): 19, (3,3): 20, (3,4): 21, (3,5): 22, (3,6): 23, (3,7): 24, (3,8): 25, (3,9): 26, (3,10): 27, (3,11):
      (4,1): 25, (4,2): 26, (4,3): 27, (4,4): 28, (4,5): 29, (4,6): 30, (4,7): 31, (4,8): 32, (4,9): 33, (4,10): 34, (4,11):
      (5,1): 22, (5,2): 23, (5,3): 24, (5,4): 25, (5,5): 26, (5,6): 27, (5,7): 28, (5,8): 29, (5,9): 30, (5,10): 31, (5,11):
      (6,1): 30, (6,2): 31, (6,3): 32, (6,4): 33, (6,5): 34, (6,6): 35, (6,7): 36, (6,8): 37, (6,9): 38, (6,10): 39, (6,11):
})  # Variable costs (sample data)
model.P_min = pyo.Param(model.I, initialize={1: 50, 2: 40, 3: 30, 4: 60, 5: 55, 6: 65})  # Minimum power output (sample data)
model.P_max = pyo.Param(model.I, initialize={1: 500, 2: 600, 3: 550, 4: 700, 5: 650, 6: 750})  # Maximum power output (sample
model.R_up = pyo.Param(model.I, initialize={1: 100, 2: 120, 3: 110, 4: 130, 5: 125, 6: 140})  # Maximum power increment (samp
model.R_down = pyo.Param(model.I, initialize={1: 90, 2: 110, 3: 100, 4: 120, 5: 115, 6: 130})  # Maximum power decrement (sam
model.D = pyo.Param(model.T, initialize={1: 280, 2: 327, 3: 400, 4: 388, 5: 501, 6: 600, 7: 800, 8: 927, 9: 705, 10: 502, 11:

# Define variables
model.p = pyo.Var(model.I, model.T, within=pyo.NonNegativeReals)  # Power output
model.u = pyo.Var(model.I, model.T, within=pyo.Binary)  # Unit status
model.v = pyo.Var(model.I, model.T, within=pyo.Binary)  # Startup status
model.w = pyo.Var(model.I, model.T, within=pyo.Binary)  # Shutdown status

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

# Define constraints
def power_output_bounds_lower(model, i, t):
    return model.P_min[i] * model.u[i, t] <= model.p[i, t]
model.power_output_bounds_lower = pyo.Constraint(model.I, model.T, rule=power_output_bounds_lower)

def power_output_bounds_upper(model, i, t):
    return model.p[i, t] <= model.P_max[i] * model.u[i, t]
model.power_output_bounds_upper = pyo.Constraint(model.I, model.T, rule=power_output_bounds_upper)

def ramping_up(model, i, t):
    if t > 1:
        return model.p[i, t] - model.p[i, t-1] <= model.R_up[i]
    else:
        return pyo.Constraint.Skip
model.ramping_up = pyo.Constraint(model.I, model.T, rule=ramping_up)

def ramping_down(model, i, t):
    if t > 1:
        return model.p[i, t-1] - model.p[i, t] <= model.R_down[i]
    else:
        return pyo.Constraint.Skip
model.ramping_down = pyo.Constraint(model.I, model.T, rule=ramping_down)

def logical_status(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_status = pyo.Constraint(model.I, model.T, rule=logical_status)

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

def demand_satisfaction(model, t):
    return sum(model.p[i, t] for i in model.I) == model.D[t]
model.demand_satisfaction = pyo.Constraint(model.T, rule=demand_satisfaction)

def excess_capacity(model, t):
    return sum(model.P_max[i] * model.u[i, t] for i in model.I) >= 1.1 * model.D[t]
model.excess_capacity = pyo.Constraint(model.T, rule=excess_capacity)

# Solve the model
```

```
solver = pyo.SolverFactory('gurobi')
results = solver.solve(model)

# Print the results
print("Optimal Objective Value:", pyo.value(model.obj))
for i in model.I:
    for t in model.T:
        print(f"Unit {i} at time {t}: Power = {model.p[i, t].value}, Status = {model.u[i, t].value}")
```

```
Optimal Objective Value: 337814.9999999987
Unit 1 at time 1: Power = 0.0, Status = -0.0
Unit 1 at time 2: Power = 0.0, Status = -0.0
Unit 1 at time 3: Power = 0.0, Status = -0.0
Unit 1 at time 4: Power = 0.0, Status = -0.0
Unit 1 at time 5: Power = 0.0, Status = -0.0
Unit 1 at time 6: Power = 0.0, Status = -0.0
Unit 1 at time 7: Power = 0.0, Status = -0.0
Unit 1 at time 8: Power = 0.0, Status = -0.0
Unit 1 at time 9: Power = 0.0, Status = -0.0
Unit 1 at time 10: Power = 0.0, Status = -0.0
Unit 1 at time 11: Power = 0.0, Status = 0.0
Unit 1 at time 12: Power = 0.0, Status = -0.0
Unit 1 at time 13: Power = 0.0, Status = -0.0
Unit 1 at time 14: Power = 0.0, Status = -0.0
Unit 1 at time 15: Power = 0.0, Status = -0.0
Unit 2 at time 1: Power = 280.0, Status = 1.0
Unit 2 at time 2: Power = 327.0, Status = 1.0
Unit 2 at time 3: Power = 400.0, Status = 1.0
Unit 2 at time 4: Power = 388.0, Status = 1.0
Unit 2 at time 5: Power = 501.0, Status = 1.0
Unit 2 at time 6: Power = 510.00000000001296, Status = 1.0
Unit 2 at time 7: Power = 600.0, Status = 1.0
Unit 2 at time 8: Power = 600.0, Status = 1.0
Unit 2 at time 9: Power = 526.0000000000106, Status = 1.0
Unit 2 at time 10: Power = 416.0, Status = 1.0
Unit 2 at time 11: Power = 530.0000000000068, Status = 1.0
Unit 2 at time 12: Power = 600.0, Status = 1.0
Unit 2 at time 13: Power = 600.0, Status = 1.0
Unit 2 at time 14: Power = 600.0, Status = 1.0
Unit 2 at time 15: Power = 600.0, Status = 1.0
Unit 3 at time 1: Power = 0.0, Status = -0.0
Unit 3 at time 2: Power = 0.0, Status = -0.0
Unit 3 at time 3: Power = 0.0, Status = -0.0
Unit 3 at time 4: Power = 0.0, Status = -0.0
Unit 3 at time 5: Power = 0.0, Status = 0.0
Unit 3 at time 6: Power = 89.99999999998704, Status = 1.0
Unit 3 at time 7: Power = 199.9999999999871, Status = 1.0
Unit 3 at time 8: Power = 272.0, Status = 1.0
Unit 3 at time 9: Power = 178.9999999999751, Status = 1.0
Unit 3 at time 10: Power = 85.99999999999997, Status = 1.0
Unit 3 at time 11: Power = 195.99999999999318, Status = 1.0
Unit 3 at time 12: Power = 305.9999999999932, Status = 1.0
Unit 3 at time 13: Power = 329.99999999999534, Status = 1.0
Unit 3 at time 14: Power = 276.99999999999613, Status = 1.0
Unit 3 at time 15: Power = 365.9999999999962, Status = 1.0
Unit 4 at time 1: Power = 0.0, Status = -0.0
Unit 4 at time 2: Power = 0.0, Status = -0.0
Unit 4 at time 3: Power = 0.0, Status = -0.0
Unit 4 at time 4: Power = 0.0, Status = -0.0
Unit 4 at time 5: Power = 0.0, Status = -0.0
Unit 4 at time 6: Power = 0.0, Status = -0.0
Unit 4 at time 7: Power = 0.0, Status = -0.0
Unit 4 at time 8: Power = 0.0, Status = -0.0
Unit 4 at time 9: Power = 0.0, Status = -0.0
Unit 4 at time 10: Power = 0.0, Status = -0.0
Unit 4 at time 11: Power = 0.0, Status = -0.0
Unit 4 at time 12: Power = 0.0, Status = -0.0
```

```
# Display results
print(f'Total costs are: {model.obj()}')
print()
for t in model.T:
    print(f"Time Period {t}:")
    for i in model.I:
        print(f"  {i}: Output={model.p[i, t].value}, Running={model.u[i, t].value}, Startup={model.v[i, t].value}, Shutdown=
```

```
Total costs are: 337814.9999999987

Time Period 1:
  1: Output=0.0, Running=-0.0, Startup=0.0, Shutdown=0.0
  2: Output=280.0, Running=1.0, Startup=0.0, Shutdown=0.0
  3: Output=0.0, Running=-0.0, Startup=0.0, Shutdown=0.0
  4: Output=0.0, Running=-0.0, Startup=0.0, Shutdown=0.0
  5: Output=0.0, Running=-0.0, Startup=0.0, Shutdown=0.0
  6: Output=0.0, Running=-0.0, Startup=0.0, Shutdown=0.0
Time Period 2:
  1: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  2: Output=327.0, Running=1.0, Startup=-0.0, Shutdown=-0.0
```

```
  3: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  4: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  5: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  6: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
Time Period 3:
  1: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  2: Output=400.0, Running=1.0, Startup=-0.0, Shutdown=-0.0
  3: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  4: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  5: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  6: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
Time Period 4:
  1: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  2: Output=388.0, Running=1.0, Startup=-0.0, Shutdown=-0.0
  3: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  4: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  5: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  6: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
Time Period 5:
  1: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  2: Output=501.0, Running=1.0, Startup=-0.0, Shutdown=-0.0
  3: Output=0.0, Running=0.0, Startup=0.0, Shutdown=-0.0
  4: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  5: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  6: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
Time Period 6:
  1: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  2: Output=510.00000000001296, Running=1.0, Startup=-0.0, Shutdown=-0.0
  3: Output=89.99999999998704, Running=1.0, Startup=1.0, Shutdown=-0.0
  4: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  5: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  6: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
Time Period 7:
  1: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  2: Output=600.0, Running=1.0, Startup=-0.0, Shutdown=-0.0
  3: Output=199.9999999999871, Running=1.0, Startup=-0.0, Shutdown=-0.0
  4: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  5: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  6: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
Time Period 8:
  1: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  2: Output=600.0, Running=1.0, Startup=-0.0, Shutdown=-0.0
  3: Output=272.0, Running=1.0, Startup=-0.0, Shutdown=-0.0
  4: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
  5: Output=55.0, Running=1.0, Startup=1.0, Shutdown=-0.0
  6: Output=0.0, Running=-0.0, Startup=-0.0, Shutdown=-0.0
```