

0. Imports and Setting up Anthropic API Client

```
from google.colab import drive
```

```
drive.mount('/content/drive')
```

↗ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)

```
!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 (Objective):\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 (Objective):
 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 the unit commitment problem you described:

Binary decision variables:

$u[i,t]$ – 1 if unit i is online (committed) in period t , 0 otherwise
 $v[i,t]$ – 1 if unit i is started up at the beginning of period t , 0 otherwise
 $w[i,t]$ – 1 if unit i is shut down at the beginning of period t , 0 otherwise

Continuous decision variables:

$p[i,t]$ – power output of unit i in period t

Parameters:

$SC[i]$ – startup cost for unit i
 $SD[i]$ – shutdown cost for unit i
 $FC[i]$ – fixed cost for unit i (when online)
 $VC[i,t]$ – variable cost for unit i in period t
 $Pmin[i]$ – minimum power output for unit i when online
 $Pmax[i]$ – maximum power output for unit i when online
 $RU[i]$ – maximum ramp-up rate for unit i (power increment limit)
 $RD[i]$ – maximum ramp-down rate for unit i (power decrement limit)
 $D[t]$ – power demand in period t

Indices:

i – indexes the thermal units ($i = 1, \dots, 6$)
 t – indexes the time periods ($t = 1, \dots, 15$)

The objective is to minimize the total cost, which consists of the startup costs, shutdown costs, fixed costs and variable costs.
 – Demand must be met in each period
 – Power output is limited by the unit's capacities and ramp rates
 – Relationship between the unit commitment status and startup/shutdown events
 – Required spinning reserve margin of 10% above demand

Let me know if you need me to formulate the full mixed-integer programming model!

✓ 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)
```

Here is the objective function for this unit commitment problem:

Minimize:

$$\sum_{i=1}^6 \sum_{t=1}^{15} (SC[i] * v[i,t] + SD[i] * w[i,t] + FC[i] * u[i,t] + VC[i,t] * p[i,t])$$

The objective is to minimize the total cost over all units i and periods t , where the total cost consists of:

- Startup costs: $SC[i] * v[i,t]$
Incurred when unit i is started up at the beginning of period t ($v[i,t]=1$)
- Shutdown costs: $SD[i] * w[i,t]$
Incurred when unit i is shut down at the beginning of period t ($w[i,t]=1$)
- Fixed costs: $FC[i] * u[i,t]$
Incurred in each period t that unit i is online ($u[i,t]=1$)
- Variable costs: $VC[i,t] * p[i,t]$
Depend on the power output $p[i,t]$ of unit i in period t

The objective function sums up all these costs over all units and periods to obtain the total cost to be minimized. Of c

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)
```

Here are the constraints for this unit commitment problem:

1. Demand constraint:
 $\sum_{i=1}^6 p[i,t] = D[t]$, for $t = 1, \dots, 15$
 The total power generated by all units in each period must equal the demand.
2. Spinning reserve constraint:
 $\sum_{i=1}^6 u[i,t] * P_{max}[i] \geq 1.1 * D[t]$, for $t = 1, \dots, 15$
 The total maximum capacity of online units must be at least 10% higher than the demand in each period for security.
3. Generation limits:
 $P_{min}[i] * u[i,t] \leq p[i,t] \leq P_{max}[i] * u[i,t]$, for $i = 1, \dots, 6$; $t = 1, \dots, 15$
 If unit i is online in period t , its output must be between its min and max capacity. If offline, output is 0.
4. Ramp-up constraints:
 $p[i,t] - p[i,t-1] \leq RU[i] * u[i,t-1] + P_{max}[i] * v[i,t]$, for $i = 1, \dots, 6$; $t = 2, \dots, 15$
 The increase in a unit's output between periods is constrained by its ramp-up limit if it was online, or its max capacity if it was started up.
5. Ramp-down constraints:
 $p[i,t-1] - p[i,t] \leq RD[i] * u[i,t] + P_{max}[i] * w[i,t]$, for $i = 1, \dots, 6$; $t = 2, \dots, 15$
 The decrease in a unit's output between periods is constrained by its ramp-down limit if it stays online, or its max capacity if it was shut down.
6. Logical constraint 1:
 $v[i,t] - w[i,t] = u[i,t] - u[i,t-1]$, for $i = 1, \dots, 6$; $t = 2, \dots, 15$
 $v[i,1] - w[i,1] = u[i,1]$, for $i = 1, \dots, 6$
 Relates the unit commitment variables to the startup and shutdown variables.
7. Logical constraint 2:
 $v[i,t] + w[i,t] \leq 1$, for $i = 1, \dots, 6$; $t = 1, \dots, 15$
 A unit cannot start up and shut down in the same period.
8. Variable domains:
 $u[i,t], v[i,t], w[i,t] \in \{0, 1\}$, for $i = 1, \dots, 6$; $t = 1, \dots, 15$
 $p[i,t] \geq 0$, for $i = 1, \dots, 6$; $t = 1, \dots, 15$
 Defining the binary and continuous variables.

These constraints, together with the objective function, form a complete mixed-integer programming formulation of the un

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 (Objective): 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)

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

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

# Define constraints
def demand_rule(model, t):
    return sum(model.p[i,t] for i in model.I) == D[t]
model.demand_cons = pyo.Constraint(model.T, rule=demand_rule)

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

def generation_rule(model, i, t):
    return pyo.inequality(Pmin[i]*model.u[i,t], model.p[i,t], Pmax[i]*model.u[i,t])
model.generation_cons = pyo.Constraint(model.I, model.T, rule=generation_rule)

def rampup_rule(model, i, t):
    if t > 1:
        return model.p[i,t] - model.p[i,t-1] <= RU[i]*model.u[i,t-1] + Pmax[i]*model.v[i,t]
    else:
        return pyo.Constraint.Skip
model.rampup_cons = pyo.Constraint(model.I, model.T, rule=rampup_rule)

def rampdown_rule(model, i, t):
    if t > 1:
        return model.p[i,t-1] - model.p[i,t] <= RD[i]*model.u[i,t] + Pmax[i]*model.w[i,t]
    else:
        return pyo.Constraint.Skip
model.rampdown_cons = pyo.Constraint(model.I, model.T, rule=rampdown_rule)

def logical_rule1(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 model.v[i,t] - model.w[i,t] == model.u[i,t]
model.logical_cons1 = pyo.Constraint(model.I, model.T, rule=logical_rule1)

def logical_rule2(model, i, t):
    return model.v[i,t] + model.w[i,t] <= 1
model.logical_cons2 = pyo.Constraint(model.I, model.T, rule=logical_rule2)

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

# Print the results
print(results)
\\`
```

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

# Create a model
model = pyo.ConcreteModel()

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

# Define parameters (using sample data)
SC = {1: 10324, 2: 5678, 3: 7802, 4: 12899, 5: 4596, 6: 9076} # Startup costs
SD = {1: 2673, 2: 5893, 3: 982, 4: 6783, 5: 2596, 6: 3561} # Shutdown costs
FC = {1: 2000, 2: 3000, 3: 2500, 4: 4000, 5: 3500, 6: 4500} # Fixed costs
VC = {(1,1): 20, (1,2): 22, (1,3): 23, (1,4): 24, (1,5): 25, (1,6): 26, (1,7): 27, (1,8): 28, (1,9): 29, (1,10): 30, (1,11):
      (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):
}
Pmin = {1: 50, 2: 40, 3: 30, 4: 60, 5: 55, 6: 65} # Minimum power output
Pmax = {1: 500, 2: 600, 3: 550, 4: 700, 5: 650, 6: 750} # Maximum power output
RU = {1: 100, 2: 120, 3: 110, 4: 130, 5: 125, 6: 140} # Ramp-up limits
RD = {1: 90, 2: 110, 3: 100, 4: 120, 5: 115, 6: 130} # Ramp-down limits
D = {1: 280, 2: 327, 3: 400, 4: 388, 5: 501, 6: 600, 7: 800, 8: 927, 9: 705, 10: 502, 11: 781, 12: 906, 13: 930, 14: 877, 15:

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

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

# Define constraints
def demand_rule(model, t):
    return sum(model.p[i,t] for i in model.I) == D[t]
model.demand_cons = pyo.Constraint(model.T, rule=demand_rule)

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

def generation_rule(model, i, t):
    return pyo.inequality(Pmin[i]*model.u[i,t], model.p[i,t], Pmax[i]*model.u[i,t])
model.generation_cons = pyo.Constraint(model.I, model.T, rule=generation_rule)

def rampup_rule(model, i, t):
    if t > 1:
        return model.p[i,t] - model.p[i,t-1] <= RU[i]*model.u[i,t-1] + Pmax[i]*model.v[i,t]
    else:
        return pyo.Constraint.Skip
model.rampup_cons = pyo.Constraint(model.I, model.T, rule=rampup_rule)

def rampdown_rule(model, i, t):
    if t > 1:
        return model.p[i,t-1] - model.p[i,t] <= RD[i]*model.u[i,t] + Pmax[i]*model.w[i,t]
    else:
        return pyo.Constraint.Skip
model.rampdown_cons = pyo.Constraint(model.I, model.T, rule=rampdown_rule)

def logical_rule1(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 model.v[i,t] - model.w[i,t] == model.u[i,t]
model.logical_cons1 = pyo.Constraint(model.I, model.T, rule=logical_rule1)

def logical_rule2(model, i, t):
    return model.v[i,t] + model.w[i,t] <= 1
model.logical_cons2 = pyo.Constraint(model.I, model.T, rule=logical_rule2)

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

# Print the results
print(results)

```



```

-----
ValueError                                Traceback (most recent call last)
<ipython-input-24-55f87451be43> in <cell line: 79>()
    77 # Solve the model
    78 solver = pyo.SolverFactory('glpk')
--> 79 results = solver.solve(model)
    80
    81 # 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 'generation_cons[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)

```

# 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

```

```

# Create a model
model = pyo.ConcreteModel()

```

```

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

```

```

# Define parameters (using sample data)
SC = {1: 10324, 2: 5678, 3: 7802, 4: 12899, 5: 4596, 6: 9076} # Startup costs
SD = {1: 2673, 2: 5893, 3: 982, 4: 6783, 5: 2596, 6: 3561} # Shutdown costs
FC = {1: 2000, 2: 3000, 3: 2500, 4: 4000, 5: 3500, 6: 4500} # Fixed costs
VC = {(1,1): 20, (1,2): 22, (1,3): 23, (1,4): 24, (1,5): 25, (1,6): 26, (1,7): 27, (1,8): 28, (1,9): 29, (1,10): 30, (1,11):
      (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):
      }
Pmin = {1: 50, 2: 40, 3: 30, 4: 60, 5: 55, 6: 65} # Minimum power output
Pmax = {1: 500, 2: 600, 3: 550, 4: 700, 5: 650, 6: 750} # Maximum power output
RU = {1: 100, 2: 120, 3: 110, 4: 130, 5: 125, 6: 140} # Ramp-up limits
RD = {1: 90, 2: 110, 3: 100, 4: 120, 5: 115, 6: 130} # Ramp-down limits
D = {1: 280, 2: 327, 3: 400, 4: 388, 5: 501, 6: 600, 7: 800, 8: 927, 9: 705, 10: 502, 11: 781, 12: 906, 13: 930, 14: 877, 15:

```

```

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

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

# Define constraints
def demand_rule(model, t):
    return sum(model.p[i,t] for i in model.I) == D[t]
model.demand_cons = pyo.Constraint(model.T, rule=demand_rule)

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

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

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

def rampup_rule(model, i, t):
    if t > 1:
        return model.p[i,t] - model.p[i,t-1] <= RU[i]*model.u[i,t-1] + Pmax[i]*model.v[i,t]
    else:
        return pyo.Constraint.Skip
model.rampup_cons = pyo.Constraint(model.I, model.T, rule=rampup_rule)

def rampdown_rule(model, i, t):
    if t > 1:
        return model.p[i,t-1] - model.p[i,t] <= RD[i]*model.u[i,t] + Pmax[i]*model.w[i,t]
    else:
        return pyo.Constraint.Skip
model.rampdown_cons = pyo.Constraint(model.I, model.T, rule=rampdown_rule)

def logical_rule1(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 model.v[i,t] - model.w[i,t] == model.u[i,t]
model.logical_cons1 = pyo.Constraint(model.I, model.T, rule=logical_rule1)

def logical_rule2(model, i, t):
    return model.v[i,t] + model.w[i,t] <= 1
model.logical_cons2 = pyo.Constraint(model.I, model.T, rule=logical_rule2)

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

# Print the results
print(results)

```



```

Problem:
- Name: x1
  Lower bound: 333254.2395363814
  Upper bound: 333275.99999999999
  Number of objectives: 1
  Number of constraints: 558
  Number of variables: 360
  Number of binary variables: 270
  Number of integer variables: 270
  Number of continuous variables: 90
  Number of nonzeros: 1746
  Sense: minimize
Solver:
- Status: ok
  Return code: 0
  Message: Model was solved to optimality (subject to tolerances), and an optimal solution is available.
  Termination condition: optimal
  Termination message: Model was solved to optimality (subject to tolerances), and an optimal solution is available.
  Wall time: 0.49279093742370605
  Error rc: 0
  Time: 1.0127315521240234

```

Solution:

```
- number of solutions: 0
  number of solutions displayed: 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: 333275.99999999999
```

Time Period 1:

```
1: Output=0.0, Running=0.0, Startup=0.0, Shutdown=0.0
2: Output=280.0, Running=1.0, Startup=1.0, 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=523.0000000000041, 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=76.99999999999596, Running=1.0, Startup=1.0, 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=598.0000000000044, 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=201.99999999999554, Running=1.0, 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=599.9999999999999, Running=1.0, Startup=0.0, Shutdown=0.0
3: Output=4.433786671143025e-12, Running=0.0, Startup=0.0, Shutdown=-0.0
4: Output=0.0, Running=0.0, Startup=0.0, Shutdown=0.0
5: Output=326.99999999999557, Running=1.0, Startup=-0.0, Shutdown=-0.0
6: Output=0.0, Running=0.0, Startup=-0.0, Shutdown=0.0
```