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
```

```
Collecting anthropic
  Downloading anthropic-0.28.0-py3-none-any.whl (862 kB)
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 862.7/862.7 kB 5.6 MB/s eta 0:00:00
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-packages (from anthropic) (1.7.0)
Collecting httpx<1,>=0.23.0 (from anthropic)
  Downloading httpx-0.27.0-py3-none-any.whl (75 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 8.0 MB/s eta 0:00:00
Collecting jiter<1,>=0.4.0 (from anthropic)
  Downloading jiter-0.4.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (328 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 328.3/328.3 kB 8.5 MB/s eta 0:00:00
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (2.7.3)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from anthropic) (1.3.1)
Requirement already satisfied: tokenizers>=0.13.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (0.19.1)
Requirement already satisfied: typing-extensions<5,>=4.7 in /usr/local/lib/python3.10/dist-packages (from anthropic) (4.
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->anthropic) (3
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->anthropi
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->anthropic) (20
Collecting httpcore==1.* (from httpx<1,>=0.23.0->anthropic)
  Downloading httpcore-1.0.5-py3-none-any.whl (77 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 8.2 MB/s eta 0:00:00
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->anthropic)
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 6.6 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.
Requirement already satisfied: pydantic-core==2.18.4 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.0
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/lib/python3.10/dist-packages (from tokenizers>
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->t
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->t
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->huggi
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface
Installing collected packages: jiter, h11, httpcore, httpx, anthropic
Successfully installed anthropic-0.28.0 h11-0.14.0 httpcore-1.0.5 httpx-0.27.0 jiter-0.4.1
```

```
# 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 mathematical optimization problem of scheduling the power output of 6 electric power ther

Continuous Variables:
- p(i,t) = Power output of unit i in period t, for i = 1,...,6 and t = 1,...,15

Binary Variables:
- u(i,t) = 1 if unit i is running in period t, 0 otherwise, for i = 1,...,6 and t = 1,...,15
- v(i,t) = 1 if unit i is started up in period t, 0 otherwise, for i = 1,...,6 and t = 1,...,15
- w(i,t) = 1 if unit i is shut down in period t, 0 otherwise, for i = 1,...,6 and t = 1,...,15

Parameters:
- C_start(i) = Startup cost for unit i, for i = 1,...,6
- C_stop(i) = Shutdown cost for unit i, for i = 1,...,6
- C_fixed(i) = Fixed cost for unit i, for i = 1,...,6
- C_var(i,t) = Variable cost for unit i in period t, for i = 1,...,6 and t = 1,...,15
- P_min(i) = Minimum power output for unit i when running, for i = 1,...,6
- P_max(i) = Maximum power output for unit i when running, for i = 1,...,6
- R_up(i) = Maximum ramp-up rate for unit i, for i = 1,...,6
- R_down(i) = Maximum ramp-down rate for unit i, for i = 1,...,6
- D(t) = Power demand in period t, for t = 1,...,15
```

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 the mathematical optimization problem of scheduling the power output of 6 electric po

Minimize:
  ∑(i=1 to 6) ∑(t=1 to 15) [C_start(i)*v(i,t) + C_stop(i)*w(i,t) + C_fixed(i)*u(i,t) + C_var(i,t)*p(i,t)]

The objective is to minimize the total cost, which consists of:
- Startup costs: C_start(i)*v(i,t) is incurred whenever unit i is started up at the beginning of period t
- Shutdown costs: C_stop(i)*w(i,t) is incurred whenever unit i is shut down at the beginning of period t
- Fixed costs: C_fixed(i)*u(i,t) is incurred for each period t that unit i is running
- Variable costs: C_var(i,t)*p(i,t) depends on the power output p(i,t) of unit i in period t

The two summations over i=1 to 6 and t=1 to 15 indicate that we are summing the costs over all 6 generation units and al

Some key points about the objective:
- The startup and shutdown costs are multiplied by the binary variables v(i,t) and w(i,t) respectively. This ensures tho
- The fixed cost is multiplied by the binary variable u(i,t) to only account for it when a unit is running.
- Variable cost is proportional to the continuous power output variable p(i,t).

Let me know if you need any clarification or have additional questions!
```

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 the mathematical optimization problem of scheduling the power output of 6 electric power th

Power output limits:
P_min(i) * u(i,t) <= p(i,t) <= P_max(i) * u(i,t), for i = 1,...,6 and t = 1,...,15

Ramp-up constraints:
p(i,t) - p(i,t-1) <= R_up(i) * u(i,t-1) + P_max(i) * v(i,t), for i = 1,...,6 and t = 2,...,15

Ramp-down constraints:
p(i,t-1) - p(i,t) <= R_down(i) * u(i,t) + P_max(i) * w(i,t), for i = 1,...,6 and t = 2,...,15

Logical constraints linking u, v, and w:
u(i,t) - u(i,t-1) = v(i,t) - w(i,t), for i = 1,...,6 and t = 2,...,15
v(i,t) + w(i,t) <= 1, for i = 1,...,6 and t = 1,...,15

Demand satisfaction:
∑(i=1 to 6) p(i,t) >= D(t), for t = 1,...,15

Security constraint (10% excess capacity):
∑(i=1 to 6) P_max(i) * u(i,t) >= 1.1 * D(t), for t = 1,...,15

The constraints ensure:
- Power output is within the unit's min and max limits when it's running (u=1)
- Changes in power output between periods respect the ramp-up and ramp-down limits
- Logical consistency between the on/off (u), startup (v), and shutdown (w) variables
- Total power output meets or exceeds demand in each period
```

```
- There's always at least 10% excess capacity available for security

Binary variables:
u(i,t), v(i,t), w(i,t) ∈ {0,1}, for i = 1,...,6 and t = 1,...,15

Non-negativity of power output:
p(i,t) >= 0, for i = 1,...,6 and t = 1,...,15

Let me know if you have any other questions!
```

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)
```

```
    model.D = pyo.Param(model.T, initialize={t: 1000 + 100 * t for t in model.T})

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

    # Define objective function
    def obj_rule(model):
        return sum(model.C_start[i] * model.v[i, t] + model.C_stop[i] * model.w[i, t] +
                   model.C_fixed[i] * model.u[i, t] + model.C_var[i, t] * model.p[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_limits(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_limits = pyo.Constraint(model.I, model.T, rule=power_output_limits)

    def ramp_up_constraint(model, i, t):
        if t == 1:
            return pyo.Constraint.Skip
        return model.p[i, t] - model.p[i, t-1] <= model.R_up[i] * model.u[i, t-1] + model.P_max[i] * model.v[i, t]
```

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

    # 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

# Define the model
model = pyo.ConcreteModel()

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

# Define parameters (using sample data)
model.C_start = pyo.Param(model.I, initialize={1: 10324, 2: 5678, 3: 7802, 4: 12899, 5: 4596, 6: 9076})
model.C_stop = pyo.Param(model.I, initialize={1: 2673, 2: 5893, 3: 982, 4: 6783, 5: 2596, 6: 3561})
model.C_fixed = pyo.Param(model.I, initialize={1: 2000, 2: 3000, 3: 2500, 4: 4000, 5: 3500, 6: 4500})
model.C_var = 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):
})
model.P_min = pyo.Param(model.I, initialize={1: 50, 2: 40, 3: 30, 4: 60, 5: 55, 6: 65})
model.P_max = pyo.Param(model.I, initialize={1: 500, 2: 600, 3: 550, 4: 700, 5: 650, 6: 750})
model.R_up = pyo.Param(model.I, initialize={1: 100, 2: 120, 3: 110, 4: 130, 5: 125, 6: 140})
model.R_down = pyo.Param(model.I, initialize={1: 90, 2: 110, 3: 100, 4: 120, 5: 115, 6: 130})
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, domain=pyo.NonNegativeReals)
model.u = pyo.Var(model.I, model.T, domain=pyo.Binary)
model.v = pyo.Var(model.I, model.T, domain=pyo.Binary)
model.w = pyo.Var(model.I, model.T, domain=pyo.Binary)

# Define objective function
def obj_rule(model):
    return sum(model.C_start[i] * model.v[i, t] + model.C_stop[i] * model.w[i, t] +
               model.C_fixed[i] * model.u[i, t] + model.C_var[i, t] * model.p[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_limits(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_limits = pyo.Constraint(model.I, model.T, rule=power_output_limits)

def ramp_up_constraint(model, i, t):
    if t == 1:
        return pyo.Constraint.Skip
    return model.p[i, t] - model.p[i, t-1] <= model.R_up[i] * model.u[i, t-1] + model.P_max[i] * model.v[i, t]
model.ramp_up_constraint = pyo.Constraint(model.I, model.T, rule=ramp_up_constraint)

def ramp_down_constraint(model, i, t):
    if t == 1:
        return pyo.Constraint.Skip
    return model.p[i, t-1] - model.p[i, t] <= model.R_down[i] * model.u[i, t] + model.P_max[i] * model.w[i, t]
model.ramp_down_constraint = pyo.Constraint(model.I, model.T, rule=ramp_down_constraint)

def logical_constraint1(model, i, t):
```

```
    if t == 1:
        return pyo.Constraint.Skip
    return model.u[i, t] - model.u[i, t-1] == model.v[i, t] - model.w[i, t]
model.logical_constraint1 = pyo.Constraint(model.I, model.T, rule=logical_constraint1)

def logical_constraint2(model, i, t):
    return model.v[i, t] + model.w[i, t] <= 1
model.logical_constraint2 = pyo.Constraint(model.I, model.T, rule=logical_constraint2)

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

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

# Print the results
print(results)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-2356e4830c6c> in <cell line: 77>()
     75 # Solve the model
     76 solver = pyo.SolverFactory('glpk')
---> 77 results = solver.solve(model)
     78
     79 # 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)

```
# 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 units
model.T = pyo.RangeSet(1, 15)  # Set of time periods

# Define parameters (using sample data)
model.C_start = pyo.Param(model.I, initialize={1: 10324, 2: 5678, 3: 7802, 4: 12899, 5: 4596, 6: 9076})
model.C_stop = pyo.Param(model.I, initialize={1: 2673, 2: 5893, 3: 982, 4: 6783, 5: 2596, 6: 3561})
model.C_fixed = pyo.Param(model.I, initialize={1: 2000, 2: 3000, 3: 2500, 4: 4000, 5: 3500, 6: 4500})
model.C_var = 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):
    })
    model.P_min = pyo.Param(model.I, initialize={1: 50, 2: 40, 3: 30, 4: 60, 5: 55, 6: 65})
    model.P_max = pyo.Param(model.I, initialize={1: 500, 2: 600, 3: 550, 4: 700, 5: 650, 6: 750})
    model.R_up = pyo.Param(model.I, initialize={1: 100, 2: 120, 3: 110, 4: 130, 5: 125, 6: 140})
    model.R_down = pyo.Param(model.I, initialize={1: 90, 2: 110, 3: 100, 4: 120, 5: 115, 6: 130})
    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, domain=pyo.NonNegativeReals)
    model.u = pyo.Var(model.I, model.T, domain=pyo.Binary)
    model.v = pyo.Var(model.I, model.T, domain=pyo.Binary)
    model.w = pyo.Var(model.I, model.T, domain=pyo.Binary)

    # Define objective function
    def obj_rule(model):
        return sum(model.C_start[i] * model.v[i, t] + model.C_stop[i] * model.w[i, t] +
                   model.C_fixed[i] * model.u[i, t] + model.C_var[i, t] * model.p[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_limits_lower(model, i, t):
        return (model.P_min[i] * model.u[i, t] <= model.p[i, t])
    model.power_output_limits_lower = pyo.Constraint(model.I, model.T, rule=power_output_limits_lower)

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

    def ramp_up_constraint(model, i, t):
        if t == 1:
            return pyo.Constraint.Skip
        return model.p[i, t] - model.p[i, t-1] <= model.R_up[i] * model.u[i, t-1] + model.P_max[i] * model.v[i, t]
    model.ramp_up_constraint = pyo.Constraint(model.I, model.T, rule=ramp_up_constraint)

    def ramp_down_constraint(model, i, t):
        if t == 1:
            return pyo.Constraint.Skip
        return model.p[i, t-1] - model.p[i, t] <= model.R_down[i] * model.u[i, t] + model.P_max[i] * model.w[i, t]
    model.ramp_down_constraint = pyo.Constraint(model.I, model.T, rule=ramp_down_constraint)

    def logical_constraint1(model, i, t):
        if t == 1:
            return pyo.Constraint.Skip
        return model.u[i, t] - model.u[i, t-1] == model.v[i, t] - model.w[i, t]
    model.logical_constraint1 = pyo.Constraint(model.I, model.T, rule=logical_constraint1)

    def logical_constraint2(model, i, t):
        return model.v[i, t] + model.w[i, t] <= 1
    model.logical_constraint2 = pyo.Constraint(model.I, model.T, rule=logical_constraint2)

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

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

    # Print the results
    print(results)
```

```
    Problem:
    - Name: x1
      Lower bound: 317769.99999999965
      Upper bound: 317769.99999999965
      Number of objectives: 1
      Number of constraints: 552
      Number of variables: 360
      Number of binary variables: 270
      Number of integer variables: 270
      Number of continuous variables: 90
      Number of nonzeros: 1728
      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: 0.10041999816894531
  Error rc: 0
  Time: 0.9260151386260986
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: 317769.9999999997

Time Period 1:
  1: Output=0.0, Running=-0.0, Startup=0.0, Shutdown=0.0
  2: Output=279.99999999999693, 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=326.99999999999693, 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=399.99999999999744, 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=387.99999999999744, 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=500.9999999999963, 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=493.0000000000039, Running=1.0, Startup=-0.0, Shutdown=-0.0
  3: Output=106.99999999999608, 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=583.0000000000032, Running=1.0, Startup=-0.0, Shutdown=-0.0
  3: Output=216.99999999999676, 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=600.0000000000032, Running=1.0, Startup=-0.0, Shutdown=-0.0
  3: Output=326.99999999999676, 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
```