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/MIP2.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 the production planner in a food and beverage production fascility. You are tasked with planning production for
```

```
!pip install anthropic
```

```
Collecting anthropic
  Downloading anthropic-0.28.0-py3-none-any.whl (862 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 862.7/862.7 kB 9.6 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 9.4 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 32.3 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 10.2 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 8.3 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
```

```python
# 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

```python
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
```

```python
# Print response
print(response_p11)
```

```
Let's define the following variables for this mathematical optimization problem:

Indices:
- i: product index (1: food 1, 2: food 2, 3: beverage 1, 4: beverage 2, 5: beverage 3)
- t: time period index (1 to 12)

Decision Variables:
- P[i,t]: production quantity of product i in period t (units for food, liters for beverages)
- S[i,t]: storage quantity of product i at the end of period t (units for food, liters for beverages)

Parameters:
- fc[i,t]: fixed cost of producing product i in period t (from "fixed_cost_production.csv")
- vc_prod[i,t]: variable cost of producing product i in period t (from "variable_cost_production.csv")
- vc_stor[i,t]: variable cost of storing product i in period t (from "variable_cost_storage.csv")
- d[i,t]: maximum demand for product i in period t (from "demand.csv")
- r[i,t]: revenue per unit of product i sold in period t (from "revenue.csv")
- stor_cap[i]: storage capacity for product i (580, 687, 599, 788, 294 for i = 1 to 5)
- prod_cap[i]: production capacity for product i (1080, 908, 408, 1000, 403 for i = 1 to 5)

Binary Variables:
- y[i,t]: 1 if product i is produced in period t, 0 otherwise

These variables will be used to formulate the objective function and constraints of the optimization problem, which aims
```

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 given production planning optimization problem:

Maximize:
  Σ[i=1 to 5] Σ[t=1 to 12] (r[i,t] * min(S[i,t-1] + P[i,t], d[i,t]) - fc[i,t] * y[i,t] - vc_prod[i,t] * P[i,t] - vc_stor

Explanation:
- The objective is to maximize the total profit over the 12 time periods for all 5 products.
- For each product i and time period t:
  - Revenue is calculated as the unit revenue (r[i,t]) multiplied by the quantity sold, which is the minimum of availabl
  - Fixed cost (fc[i,t]) is incurred if production occurs (y[i,t] = 1).
  - Variable production cost (vc_prod[i,t]) is incurred for each unit produced (P[i,t]).
  - Variable storage cost (vc_stor[i,t]) is incurred for each unit stored at the end of the period (S[i,t]).
- The summation is taken over all products (i=1 to 5) and all time periods (t=1 to 12).

Note:
- The binary variable y[i,t] is used to model the fixed cost, which is only incurred if production occurs (P[i,t] > 0).
- The min(S[i,t-1] + P[i,t], d[i,t]) term ensures that the quantity sold does not exceed the demand for each product and
- S[i,0] (initial storage) is assumed to be zero for all products.

The objective function represents the total profit to be maximized, considering revenues, fixed and variable production
```

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 production planning optimization problem:

1. Production capacity constraint:
   P[i,t] ≤ prod_cap[i] * y[i,t], for all i and t

2. Storage capacity constraint:
   S[i,t] ≤ stor_cap[i], for all i and t

3. Inventory balance constraint:
   S[i,t] = S[i,t-1] + P[i,t] - min(S[i,t-1] + P[i,t], d[i,t]), for all i and t

4. Non-negativity constraints:
   P[i,t] ≥ 0 and integer, for i = 1, 2 and all t
   P[i,t] ≥ 0, for i = 3, 4, 5 and all t
   S[i,t] ≥ 0 and integer, for i = 1, 2 and all t
   S[i,t] ≥ 0, for i = 3, 4, 5 and all t

5. Binary constraint:
   y[i,t] ∈ {0, 1}, for all i and t

Explanation:
1. The production capacity constraint ensures that the production quantity (P[i,t]) does not exceed the production capac
```

```
2. The storage capacity constraint ensures that the storage quantity (S[i,t]) does not exceed the storage capacity (stor

3. The inventory balance constraint ensures that the storage quantity at the end of period t (S[i,t]) equals the storage

4. The non-negativity constraints ensure that the production quantities (P[i,t]) and storage quantities (S[i,t]) are non

5. The binary constraint ensures that the decision variable y[i,t] can only take the values 0 or 1, indicating whether p

These constraints, along with the objective function, complete the formulation of the production planning optimization p
```

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)
```

```
            [illegible]
            model.fc[i, t] * model.y[i, t] -
            model.vc_prod[i, t] * model.P[i, t] -
            model.vc_stor[i, t] * model.S[i, t]
            for i in model.I for t in model.T
        )
    model.objective = pyo.Objective(rule=objective_rule, sense=pyo.maximize)
```

```
            print(f"Product {i}, Period {t}: {model.S[i, t].value}")
    ```

    Note: Make sure to have the following CSV files in the same directory as the Python script:
    - fixed_cost_production.csv
    - variable_cost_production.csv
    - variable_cost_storage.csv
    - demand.csv
    - revenue.csv

    The CSV files should have the following structure:
    - Rows: Product index (1 to 5)
```

## 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
import pandas as pd

# Read data from CSV files
# Read data from CSV files ADJUSTED THE DATA LOADS TO WORK
fixed_cost_production = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/fixed_cost_production.csv")
fixed_cost_production.index += 1
fixed_cost_production = fixed_cost_production.drop("Unnamed: 0", axis = 1)
fixed_cost_production.columns = fixed_cost_production.columns.astype(int)

variable_cost_production = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/variable_cost_production.csv")
variable_cost_production.index += 1
variable_cost_production = variable_cost_production.drop("Unnamed: 0", axis = 1)
variable_cost_production.columns = variable_cost_production.columns.astype(int)

variable_cost_storage = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/variable_cost_storage.csv")
variable_cost_storage.index += 1
variable_cost_storage = variable_cost_storage.drop("Unnamed: 0", axis = 1)
variable_cost_storage.columns = variable_cost_storage.columns.astype(int)

demand = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/demand.csv")
demand.index += 1
demand = demand.drop("Unnamed: 0", axis = 1)
demand.columns = demand.columns.astype(int)

revenue = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/revenue.csv")
revenue.index += 1
revenue = revenue.drop("Unnamed: 0", axis = 1)
revenue.columns = revenue.columns.astype(int)

# Define model
model = pyo.ConcreteModel()

# Define sets
model.I = pyo.RangeSet(1, 5)  # Products
model.T = pyo.RangeSet(1, 12)  # Time periods

# Define parameters
model.fc = pyo.Param(model.I, model.T, initialize=fixed_cost_production.stack().to_dict())
model.vc_prod = pyo.Param(model.I, model.T, initialize=variable_cost_production.stack().to_dict())
model.vc_stor = pyo.Param(model.I, model.T, initialize=variable_cost_storage.stack().to_dict())
model.d = pyo.Param(model.I, model.T, initialize=demand.stack().to_dict())
model.r = pyo.Param(model.I, model.T, initialize=revenue.stack().to_dict())
model.stor_cap = pyo.Param(model.I, initialize={1: 580, 2: 687, 3: 599, 4: 788, 5: 294})
model.prod_cap = pyo.Param(model.I, initialize={1: 1080, 2: 908, 3: 408, 4: 1000, 5: 403})

# Define variables
model.P = pyo.Var(model.I, model.T, domain=pyo.NonNegativeReals)
model.S = pyo.Var(model.I, model.T, domain=pyo.NonNegativeReals)
model.y = pyo.Var(model.I, model.T, domain=pyo.Binary)

# Define objective function
def objective_rule(model):
    return sum(
        model.r[i, t] * min(model.S[i, t-1] + model.P[i, t], model.d[i, t]) -
        model.fc[i, t] * model.y[i, t] -
        model.vc_prod[i, t] * model.P[i, t] -
        model.vc_stor[i, t] * model.S[i, t]
        for i in model.I for t in model.T
    )
model.objective = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

# Define constraints
def production_capacity_rule(model, i, t):
    return model.P[i, t] <= model.prod_cap[i] * model.y[i, t]
model.production_capacity = pyo.Constraint(model.I, model.T, rule=production_capacity_rule)

def storage_capacity_rule(model, i, t):
    return model.S[i, t] <= model.stor_cap[i]
model.storage_capacity = pyo.Constraint(model.I, model.T, rule=storage_capacity_rule)

def inventory_balance_rule(model, i, t):
    if t == 1:
        return model.S[i, t] == model.P[i, t] - min(model.P[i, t], model.d[i, t])
    else:
        return model.S[i, t] == model.S[i, t-1] + model.P[i, t] - min(model.S[i, t-1] + model.P[i, t], model.d[i, t])
model.inventory_balance = pyo.Constraint(model.I, model.T, rule=inventory_balance_rule)

def integrality_rule(model, i, t):
    if i <= 2:
        return model.P[i, t].is_integer() and model.S[i, t].is_integer()
    else:
        return pyo.Constraint.Skip
model.integrality = pyo.Constraint(model.I, model.T, rule=integrality_rule)

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

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

print("\nProduction quantities:")
for i in model.I:
    for t in model.T:
        print(f"Product {i}, Period {t}: {model.P[i, t].value}")

print("\nStorage quantities:")
for i in model.I:
    for t in model.T:
        print(f"Product {i}, Period {t}: {model.S[i, t].value}")
```

```
ERROR:pyomo.core:Rule failed when generating expression for Objective objecti
KeyError: "Index '(1, 0)' is not valid for indexed component 'S'"
ERROR:pyomo.core:Constructing component 'objective' from data=None failed:
    KeyError: "Index '(1, 0)' is not valid for indexed component 'S'"
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-15-76a536f40b52> in <cell line: 61>()
     59         for i in model.I for t in model.T
     60     )
---> 61 model.objective = pyo.Objective(rule=objective_rule,
sense=pyo.maximize)
     62 
     63 # Define constraints

                              8 frames
/usr/local/lib/python3.10/dist-packages/pyomo/core/base/indexed_component.py
in _validate_index(self, idx)
    864         # Raise an exception
    865         #
--> 866         raise KeyError(
    867             "Index '%s' is not valid for indexed component '%s'"
    868             % (normalized_idx, self.name)
```

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
import pandas as pd

# Read data from CSV files ADJUSTED THE DATA LOADS TO WORK
fixed_cost_production = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/fixed_cost_production.csv")
fixed_cost_production.index += 1
fixed_cost_production = fixed_cost_production.drop("Unnamed: 0", axis = 1)
fixed_cost_production.columns = fixed_cost_production.columns.astype(int)

variable_cost_production = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/variable_cost_production.csv")
variable_cost_production.index += 1
variable_cost_production = variable_cost_production.drop("Unnamed: 0", axis = 1)
variable_cost_production.columns = variable_cost_production.columns.astype(int)

variable_cost_storage = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/variable_cost_storage.csv")
```

```
    variable_cost_storage = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/variable_cost_storage.csv")
    variable_cost_storage.index += 1
    variable_cost_storage = variable_cost_storage.drop("Unnamed: 0", axis = 1)
    variable_cost_storage.columns = variable_cost_storage.columns.astype(int)

    demand = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/demand.csv")
    demand.index += 1
    demand = demand.drop("Unnamed: 0", axis = 1)
    demand.columns = demand.columns.astype(int)

    revenue = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/revenue.csv")
    revenue.index += 1
    revenue = revenue.drop("Unnamed: 0", axis = 1)
    revenue.columns = revenue.columns.astype(int)

    # Define model
    model = pyo.ConcreteModel()

    # Define sets
    model.I = pyo.RangeSet(1, 5)  # Products
    model.T = pyo.RangeSet(1, 12)  # Time periods

    # Define parameters
    model.fc = pyo.Param(model.I, model.T, initialize=fixed_cost_production.stack().to_dict())
    model.vc_prod = pyo.Param(model.I, model.T, initialize=variable_cost_production.stack().to_dict())
    model.vc_stor = pyo.Param(model.I, model.T, initialize=variable_cost_storage.stack().to_dict())
    model.d = pyo.Param(model.I, model.T, initialize=demand.stack().to_dict())
    model.r = pyo.Param(model.I, model.T, initialize=revenue.stack().to_dict())
    model.stor_cap = pyo.Param(model.I, initialize={1: 580, 2: 687, 3: 599, 4: 788, 5: 294})
    model.prod_cap = pyo.Param(model.I, initialize={1: 1080, 2: 908, 3: 408, 4: 1000, 5: 403})

    # Define variables
    model.P = pyo.Var(model.I, model.T, domain=pyo.NonNegativeReals)
    model.S = pyo.Var(model.I, model.T, domain=pyo.NonNegativeReals)
    model.y = pyo.Var(model.I, model.T, domain=pyo.Binary)
    model.m = pyo.Var(model.I, model.T, domain=pyo.NonNegativeReals)  # Auxiliary variable for min(x + s[t-1], d)

    #set domain to integer for food
    for t in model.T:
        model.P[1, t].domain = pyo.NonNegativeIntegers
        model.P[2, t].domain = pyo.NonNegativeIntegers
        model.S[1, t].domain = pyo.NonNegativeIntegers
        model.S[2, t].domain = pyo.NonNegativeIntegers
        model.m[1, t].domain = pyo.NonNegativeIntegers
        model.m[2, t].domain = pyo.NonNegativeIntegers

    # Define objective function
    def objective_rule(model):
        return sum(
            model.r[i, t] * model.m[i, t] -
            model.fc[i, t] * model.y[i, t] -
            model.vc_prod[i, t] * model.P[i, t] -
            model.vc_stor[i, t] * model.S[i, t]
            for i in model.I for t in model.T
        )
    model.objective = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

    # Define constraints
    def production_capacity_rule(model, i, t):
        return model.P[i, t] <= model.prod_cap[i] * model.y[i, t]
    model.production_capacity = pyo.Constraint(model.I, model.T, rule=production_capacity_rule)

    def storage_capacity_rule(model, i, t):
        return model.S[i, t] <= model.stor_cap[i]
    model.storage_capacity = pyo.Constraint(model.I, model.T, rule=storage_capacity_rule)

    def inventory_balance_rule(model, i, t):
        if t == 1:
            return model.S[i, t] == model.P[i, t] - model.m[i, t]
        else:
            return model.S[i, t] == model.S[i, t-1] + model.P[i, t] - model.m[i, t]
    model.inventory_balance = pyo.Constraint(model.I, model.T, rule=inventory_balance_rule)

    # Auxiliary constraints
    def min_demand_constraint(model, i, t):
        if t == 1:
            return model.m[i, t] <= model.P[i, t] + 0
        else:
            return model.m[i, t] <= model.P[i, t] + model.S[i, t-1]
    model.min_demand_constraint = pyo.Constraint(model.I, model.T, rule=min_demand_constraint)

    def min_demand_constraint_demand(model, i, t):
        return model.m[i, t] <= model.d[i, t]
```

```
model.min_demand_constraint_demand = pyo.Constraint(model.I, model.T, rule=min_demand_constraint_demand)

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

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

print("\nProduction quantities:")
for i in model.I:
    for t in model.T:
        print(f"Product {i}, Period {t}: {model.P[i, t].value}")

print("\nStorage quantities:")
for i in model.I:
    for t in model.T:
        print(f"Product {i}, Period {t}: {model.S[i, t].value}")
```

```
Objective value: 4523.289299999999

Production quantities:
Product 1, Period 1: -0.0
Product 1, Period 2: -0.0
Product 1, Period 3: -0.0
Product 1, Period 4: -0.0
Product 1, Period 5: -0.0
Product 1, Period 6: -0.0
Product 1, Period 7: -0.0
Product 1, Period 8: -0.0
Product 1, Period 9: -0.0
Product 1, Period 10: -0.0
Product 1, Period 11: -0.0
Product 1, Period 12: -0.0
Product 2, Period 1: -0.0
Product 2, Period 2: -0.0
Product 2, Period 3: -0.0
```