

## ✓ 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 (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 the production planner in a food and beverage production facility. You are tasked with planning production for

```
!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)
```

→ To formulate the variables for this mathematical optimization problem, let's define the following:

Indices:

- i: product index (i = 1, 2 for food; i = 3, 4, 5 for beverages)
- t: time period index (t = 1, 2, ..., 12)

Decision variables:

- x[i,t]: production quantity of product i in period t (integer for food, continuous for beverages)
- s[i,t]: storage quantity of product i at the end of period t (integer for food, continuous for beverages)
- y[i,t]: binary variable indicating whether product i is produced in period t (1 if produced, 0 otherwise)

Input parameters:

- f[i,t]: fixed cost of producing product i in period t
- cp[i,t]: variable cost of producing one unit of product i in period t
- cs[i,t]: variable cost of storing one unit of product i in period t
- d[i,t]: maximum demand for product i in period t
- r[i,t]: revenue per unit of product i sold in period t
- SC[i]: storage capacity for product i (580, 687, 599, 788, 294)
- PC[i]: production capacity for product i (1080, 908, 408, 1000, 403)

The input parameters f[i,t], cp[i,t], cs[i,t], d[i,t], and r[i,t] are provided in the respective CSV files.

Please note that the decision variables x[i,t] and s[i,t] are defined as integer variables for food products (i = 1, 2)

## ✓ 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 information provided, the objective function for this mathematical optimization problem can be formulated a

Maximize:

$$\sum_{t=1}^{12} \sum_{i=1}^5 (r[i,t] * \min(x[i,t] + s[i,t-1], d[i,t]) - f[i,t] * y[i,t] - cp[i,t] * x[i,t] - cs[i,t] * s[i,t])$$

where:

- $\min(x[i,t] + s[i,t-1], d[i,t])$  represents the actual sales of product i in period t, considering the available quantit

- $r[i,t] * \min(x[i,t] + s[i,t-1], d[i,t])$  represents the revenue from selling product  $i$  in period  $t$ .
- $f[i,t] * y[i,t]$  represents the fixed cost of production for product  $i$  in period  $t$ , where  $y[i,t]$  is a binary variable  $i$
- $cp[i,t] * x[i,t]$  represents the variable cost of production for product  $i$  in period  $t$ .
- $cs[i,t] * s[i,t]$  represents the variable cost of storage for product  $i$  at the end of period  $t$ .

The objective function aims to maximize the total profit over all time periods and products by subtracting the fixed cos

Please note that  $s[i,0]$  (storage at the beginning of the planning horizon) is assumed to be 0 for all products.

## ✓ 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 the detailed problem description. I will now formulate the constraints for this mathematical optimization

Constraints:

1. Production capacity constraints:
  - For food products ( $i = 1, 2$ ):
 
$$x[i,t] \leq PC[i] * y[i,t], \text{ for all } t$$
  - For beverage products ( $i = 3, 4, 5$ ):
 
$$x[i,t] \leq PC[i] * y[i,t], \text{ for all } t$$
2. Storage capacity constraints:
 
$$s[i,t] \leq SC[i], \text{ for all } i \text{ and } t$$
3. Storage balance constraints:
 
$$s[i,t] = s[i,t-1] + x[i,t] - \min(x[i,t] + s[i,t-1], d[i,t]), \text{ for all } i \text{ and } t > 1$$

$$s[i,1] = x[i,1] - \min(x[i,1], d[i,1]), \text{ for all } i$$
4. Non-negativity and integer constraints:
  - For food products ( $i = 1, 2$ ):
 
$$x[i,t] \geq 0 \text{ and integer, for all } t$$

$$s[i,t] \geq 0 \text{ and integer, for all } t$$
  - For beverage products ( $i = 3, 4, 5$ ):
 
$$x[i,t] \geq 0, \text{ for all } t$$

$$s[i,t] \geq 0, \text{ for all } t$$
5. Binary constraints:
 
$$y[i,t] \in \{0, 1\}, \text{ for all } i \text{ and } t$$

Explanation:

1. The production capacity constraints ensure that the production quantity of each product in each period does not exceed the capacity.
  2. The storage capacity constraints ensure that the storage quantity of each product at the end of each period does not exceed the storage capacity.
  3. The storage balance constraints define the relationship between the storage quantities across consecutive periods. For each product  $i$ , the storage at time  $t$  is equal to the storage at time  $t-1$  plus the production quantity minus the demand quantity.
  4. The non-negativity and integer constraints ensure that the production and storage quantities are non-negative. For food products, both production and storage quantities must be integers.
  5. The binary constraints define the binary nature of the  $y[i,t]$  variables, which indicate whether production occurs for product  $i$  in period  $t$ .
- These constraints, along with the objective function provided earlier, complete the formulation of the mathematical optimization model.

## ✓ 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 the sets
model.I = pyo.RangeSet(1, 5) # Product index
model.T = pyo.RangeSet(1, 12) # Time period index

# Define the parameters
model.f = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_f.iloc[t-1, i-1])
model.cp = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_cp.iloc[t-1, i-1])
model.cs = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_cs.iloc[t-1, i-1])
model.d = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_d.iloc[t-1, i-1])
model.r = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_r.iloc[t-1, i-1])

model.SC = pyo.Param(model.I, initialize={1: 580, 2: 687, 3: 599, 4: 788, 5: 294}) # Sample data for storage capacity
model.PC = pyo.Param(model.I, initialize={1: 1080, 2: 908, 3: 408, 4: 1000, 5: 403}) # Sample data for production cap

# Define the variables
model.x = pyo.Var(model.I, model.T, domain=pyo.NonNegativeIntegers)
model.s = pyo.Var(model.I, model.T, domain=pyo.NonNegativeIntegers)
model.y = pyo.Var(model.I, model.T, domain=pyo.Binary)

# Define the objective function
def obj_rule(model):
    return sum(model.r[i,t] * pyo.min(model.x[i,t] + model.s[i,t-1], model.d[i,t]) -
               model.f[i,t] * model.y[i,t] - model.cp[i,t] * model.x[i,t] - model.cs[i,t] * model.s[i,t]
               for i in model.I for t in model.T)
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)

# Define the constraints
def production_capacity_rule(model, i, t):
    return model.x[i,t] <= model.PC[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.SC[i]
model.storage_capacity = pyo.Constraint(model.I, model.T, rule=storage_capacity_rule)

def storage_balance_rule(model, i, t):
    if t == 1:
        return model.s[i,t] == model.x[i,t] - pyo.min(model.x[i,t], model.d[i,t])
    else:
        return model.s[i,t] == model.s[i,t-1] + model.x[i,t] - pyo.min(model.x[i,t] + model.s[i,t-1], model.d[i,t])
```

The sample data for storage capacity (``model.SC``) and production capacity (``model.PC``) is provided directly in the code

## 4. Input Problem Data and Test Model Code

```
# Installing pyomo and 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 input data from CSV files (replace with actual file paths)
fixed_cost_production = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/fixed_cost_production.csv")
df_f = fixed_cost_production.drop("Unnamed: 0", axis = 1).transpose()
df_f.columns = df_f.columns.astype(int)

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

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

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

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

# Define the model
model = pyo.ConcreteModel()

# Define the sets
model.I = pyo.RangeSet(1, 5) # Product index
model.T = pyo.RangeSet(1, 12) # Time period index

# Define the parameters
model.f = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_f.iloc[t-1, i-1])
model.cp = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_cp.iloc[t-1, i-1])
model.cs = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_cs.iloc[t-1, i-1])
model.d = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_d.iloc[t-1, i-1])
model.r = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_r.iloc[t-1, i-1])

model.SC = pyo.Param(model.I, initialize={1: 580, 2: 687, 3: 599, 4: 788, 5: 294}) # Sample data for storage capacity
model.PC = pyo.Param(model.I, initialize={1: 1080, 2: 908, 3: 408, 4: 1000, 5: 403}) # Sample data for production capacity

# Define the variables
model.x = pyo.Var(model.I, model.T, domain=pyo.NonNegativeIntegers)
model.s = pyo.Var(model.I, model.T, domain=pyo.NonNegativeIntegers)
model.y = pyo.Var(model.I, model.T, domain=pyo.Binary)

# Define the objective function
def obj_rule(model):
    return sum(model.r[i,t] * pyo.min(model.x[i,t] + model.s[i,t-1], model.d[i,t]) -
              model.f[i,t] * model.y[i,t] - model.cp[i,t] * model.x[i,t] - model.cs[i,t] * model.s[i,t]
              for i in model.I for t in model.T)
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)

# Define the constraints
def production_capacity_rule(model, i, t):
    return model.x[i,t] <= model.PC[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.SC[i]
model.storage_capacity = pyo.Constraint(model.I, model.T, rule=storage_capacity_rule)

def storage_balance_rule(model, i, t):
    if t == 1:
        return model.s[i,t] == model.x[i,t] - pyo.min(model.x[i,t], model.d[i,t])
    else:
        return model.s[i,t] == model.s[i,t-1] + model.x[i,t] - pyo.min(model.x[i,t] + model.s[i,t-1], model.d[i,t])
model.storage_balance = pyo.Constraint(model.I, model.T, rule=storage_balance_rule)

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

# Print the results
print(f"Objective value: {pyo.value(model.obj):.2f}")
for i in model.I:
    for t in model.T:
        print(f"Product {i}, Period {t}: Production = {pyo.value(model.x[i,t])}, Storage = {pyo.value(model.s[i,t])}")

```

```

ERROR:pyomo.core:Rule failed when generating expression for Objective obj with
AttributeError: module 'pyomo.environ' has no attribute 'min'
ERROR:pyomo.core:Constructing component 'obj' from data=None failed:
AttributeError: module 'pyomo.environ' has no attribute 'min'

```

```

-----
AttributeError                                Traceback (most recent call last)
<ipython-input-21-c421cc0f0c58> in <cell line: 52>()
    50             model.f[i,t] * model.y[i,t] - model.cp[i,t] *
model.x[i,t] - model.cs[i,t] * model.s[i,t]
    51         for i in model.I for t in model.T)
--> 52 model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)
    53
    54 # Define the constraints

```

---

```

-----
6 frames
/usr/local/lib/python3.10/dist-packages/pyomo/common/deprecation.py in
__getattr__(name)
    437         elif _mod_getattr is not None:
    438             return _mod_getattr(name)
--> 439         raise AttributeError(
    440             "module '%s' has no attribute '%s'" %
(f_globals['_name__'], name)
    441         )

```

## ✓ 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'

```

##### Linearized min functions in objective and constraints #####
##### Added correct decision variable domains #####

```

```

import pyomo.environ as pyo
import pandas as pd

```

```

# Read input data from CSV files (replace with actual file paths)
fixed_cost_production = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/MIP/MIP2/fixed_cost_production.csv")
df_f = fixed_cost_production.drop("Unnamed: 0", axis=1).transpose()
df_f.columns = df_f.columns.astype(int)

```

```

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

```

```

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

```

```

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

```

```

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

```

```

# Define the model
model = pyo.ConcreteModel()

```

```

# Define the sets
model.I = pyo.RangeSet(1, 5) # Product index
model.T = pyo.RangeSet(1, 12) # Time period index

```

```

# Define the parameters

```

```

# Define the parameters
model.f = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_f.iloc[t-1, i-1])
model.cp = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_cp.iloc[t-1, i-1])
model.cs = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_cs.iloc[t-1, i-1])
model.d = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_d.iloc[t-1, i-1])
model.r = pyo.Param(model.I, model.T, initialize=lambda model, i, t: df_r.iloc[t-1, i-1])

model.SC = pyo.Param(model.I, initialize={1: 580, 2: 687, 3: 599, 4: 788, 5: 294}) # Sample data for storage capacity
model.PC = pyo.Param(model.I, initialize={1: 1080, 2: 908, 3: 408, 4: 1000, 5: 403}) # Sample data for production capacity

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

#set domain to integer for food MODIFIED ADDED INTEGRALITY HERE INSTEAD OF CONSTRAINT
for t in model.T:
    model.x[5, t].domain = pyo.NonNegativeReals
    model.x[4, t].domain = pyo.NonNegativeReals
    model.x[3, t].domain = pyo.NonNegativeReals
    model.s[5, t].domain = pyo.NonNegativeReals
    model.s[4, t].domain = pyo.NonNegativeReals
    model.s[3, t].domain = pyo.NonNegativeReals
    model.m[5, t].domain = pyo.NonNegativeReals
    model.m[4, t].domain = pyo.NonNegativeReals
    model.m[3, t].domain = pyo.NonNegativeReals

# Define the objective function
def obj_rule(model):
    return sum(model.r[i, t] * model.m[i, t] -
              model.f[i, t] * model.y[i, t] -
              model.cp[i, t] * model.x[i, t] -
              model.cs[i, t] * model.s[i, t]
              for i in model.I for t in model.T)
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)

# Define the constraints
def production_capacity_rule(model, i, t):
    return model.x[i, t] <= model.PC[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.SC[i]
model.storage_capacity = pyo.Constraint(model.I, model.T, rule=storage_capacity_rule)

def storage_balance_rule(model, i, t):
    if t == 1:
        return model.s[i, t] == model.x[i, t] - model.m[i, t]
    else:
        return model.s[i, t] == model.s[i, t-1] + model.x[i, t] - model.m[i, t]
model.storage_balance = pyo.Constraint(model.I, model.T, rule=storage_balance_rule)

def min_demand_constraint(model, i, t):
    if t == 1:
        return model.m[i, t] <= model.x[i, t] + 0
    else:
        return model.m[i, t] <= model.x[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)

```