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
```

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)
```

```
Based on the problem description, we can define the following variables for the mathematical optimization problem:

1. Decision Variables:
   - x[i,t]: Quantity of product i produced in time period t (for food products, x[i,t] ∈ ℤ⁺; for beverage products, x[i
   - s[i,t]: Quantity of product i stored at the end of time period t (for food products, s[i,t] ∈ ℤ⁺; for beverage prod
   - y[i,t]: Binary variable indicating whether product i is produced in time period t (y[i,t] ∈ {0, 1})

   where,
   i ∈ {1, 2} for food products and i ∈ {1, 2, 3} for beverage products
   t ∈ {1, 2, ..., 12} representing the time periods

2. Parameters:
   - FC[i,t]: Fixed cost for producing product i in time period t (from "fixed_cost_production.csv")
   - VCP[i,t]: Variable cost for producing one unit of product i in time period t (from "variable_cost_production.csv")
   - VCS[i,t]: Variable cost for storing one unit of product i in time period t (from "variable_cost_storage.csv")
   - D[i,t]: Maximum demand for product i in time period t (from "demand.csv")
   - R[i,t]: Revenue per unit of product i sold in time period t (from "revenue.csv")
   - SC[i]: Storage capacity for product i (constant values provided)
   - PC[i]: Production capacity for product i (constant values provided)

These variables and parameters will be used to formulate the objective function and constraints of the mathematical opti
```

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's the formulation of the objective function for this mathematical optimization problem:

Maximize:
∑[i=1 to 2, t=1 to 12] (R[i,t] * min{s[i,t-1] + x[i,t], D[i,t]}) - ∑[i=1 to 2, t=1 to 12] (FC[i,t] * y[i,t] + VCP[i,t] *
∑[i=1 to 3, t=1 to 12] (R[i,t] * min{s[i,t-1] + x[i,t], D[i,t]}) - ∑[i=1 to 3, t=1 to 12] (FC[i,t] * y[i,t] + VCP[i,t] *

Explanation:
- The first term, ∑[i=1 to 2, t=1 to 12] (R[i,t] * min{s[i,t-1] + x[i,t], D[i,t]}), represents the total revenue from se
- The second term, ∑[i=1 to 2, t=1 to 12] (FC[i,t] * y[i,t] + VCP[i,t] * x[i,t] + VCS[i,t] * s[i,t]), represents the tot
- The third term, ∑[i=1 to 3, t=1 to 12] (R[i,t] * min{s[i,t-1] + x[i,t], D[i,t]}), represents the total revenue from se
- The fourth term, ∑[i=1 to 3, t=1 to 12] (FC[i,t] * y[i,t] + VCP[i,t] * x[i,t] + VCS[i,t] * s[i,t]), represents the tot

The objective is to maximize the difference between the total revenue and the total cost across all products and time pe

Note that this objective function assumes s[i,0] = 0 for all products i, meaning there is no initial storage at the begi
```

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:

1. Storage balance constraints:
   - For food products: s[i,t] = s[i,t-1] + x[i,t] - min{s[i,t-1] + x[i,t], D[i,t]}, ∀i ∈ {1, 2}, ∀t ∈ {1, 2, ..., 12}
   - For beverage products: s[i,t] = s[i,t-1] + x[i,t] - min{s[i,t-1] + x[i,t], D[i,t]}, ∀i ∈ {1, 2, 3}, ∀t ∈ {1, 2, ...

2. Storage capacity constraints:
   - For food products: s[i,t] ≤ SC[i], ∀i ∈ {1, 2}, ∀t ∈ {1, 2, ..., 12}
   - For beverage products: s[i,t] ≤ SC[i], ∀i ∈ {1, 2, 3}, ∀t ∈ {1, 2, ..., 12}

3. Production capacity constraints:
   - For food products: x[i,t] ≤ PC[i] * y[i,t], ∀i ∈ {1, 2}, ∀t ∈ {1, 2, ..., 12}
   - For beverage products: x[i,t] ≤ PC[i] * y[i,t], ∀i ∈ {1, 2, 3}, ∀t ∈ {1, 2, ..., 12}

4. Binary variable constraints:
   - y[i,t] ∈ {0, 1}, ∀i ∈ {1, 2, 1, 2, 3}, ∀t ∈ {1, 2, ..., 12}

5. Non-negativity constraints:
   - x[i,t] ∈ ℤ⁺, ∀i ∈ {1, 2}, ∀t ∈ {1, 2, ..., 12}
   - x[i,t] ∈ ℝ⁺, ∀i ∈ {1, 2, 3}, ∀t ∈ {1, 2, ..., 12}
   - s[i,t] ∈ ℤ⁺, ∀i ∈ {1, 2}, ∀t ∈ {1, 2, ..., 12}
   - s[i,t] ∈ ℝ⁺, ∀i ∈ {1, 2, 3}, ∀t ∈ {1, 2, ..., 12}

Explanation:
1. Storage balance constraints ensure that the storage at the end of each time period equals the storage from the previo
2. Storage capacity constraints ensure that the storage for each product does not exceed its respective storage capacity
3. Production capacity constraints ensure that the production of each product does not exceed its respective production
4. Binary variable constraints define the binary nature of the y[i,t] variables.
5. Non-negativity constraints ensure that the production and storage variables are non-negative, with integer values for

Note: The notation "∀i ∈ {1, 2}, ∀t ∈ {1, 2, ..., 12}" means "for all i in the set {1, 2} and for all t in the set {1, 2
```

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.objective = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

# Define constraints
def storage_balance_food_rule(model, i, t):
    if t == 1:
        return model.s[i, t] == model.x[i, t] - min(model.x[i, t], demand.loc[i, str(t)])
    else:
        return model.s[i, t] == model.s[i, t-1] + model.x[i, t] - min(model.s[i, t-1] + model.x[i, t], demand.loc[i, s

model.storage_balance_food = pyo.Constraint(model.food_products, model.time_periods, rule=storage_balance_food_rule)

def storage_balance_beverage_rule(model, i, t):
    if t == 1:
        return model.s_bev[i, t] == model.x_bev[i, t] - min(model.x_bev[i, t], demand.loc[i+2, str(t)])
    else:
        return model.s_bev[i, t] == model.s_bev[i, t-1] + model.x_bev[i, t] - min(model.s_bev[i, t-1] + model.x_bev[i,

model.storage_balance_beverage = pyo.Constraint(model.beverage_products, model.time_periods, rule=storage_balance_beve

def storage_capacity_food_rule(model, i, t):
    return model.s[i, t] <= storage_capacity[i]

model.storage_capacity_food = pyo.Constraint(model.food_products, model.time_periods, rule=storage_capacity_food_rule)

def storage_capacity_beverage_rule(model, i, t):
    return model.s_bev[i, t] <= storage_capacity[i+2]

model.storage_capacity_beverage = pyo.Constraint(model.beverage_products, model.time_periods, rule=storage_capacity_be

def production_capacity_food_rule(model, i, t):
    return model.x[i, t] <= production_capacity[i] * model.y[i, t]

model.production_capacity_food = pyo.Constraint(model.food_products, model.time_periods, rule=production_capacity_food

def production_capacity_beverage_rule(model, i, t):
    return model.x_bev[i, t] <= production_capacity[i+2] * model.y_bev[i, t]

model.production_capacity_beverage = pyo.Constraint(model.beverage_products, model.time_periods, rule=production_capac

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

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

print("Food products:")
for i in model.food_products:
    for t in model.time_periods:
        print(f"Product {i}, Period {t}: Production = {model.x[i, t].value}, Storage = {model.s[i, t].value}")

print("Beverage products:")
for i in model.beverage_products:
    for t in model.time_periods:
        print(f"Product {i}, Period {t}: Production = {model.x_bev[i, t].value}, Storage = {model.s_bev[i, t].value}")
```
```

## 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
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)

    # Sample data for storage and production capacities (replace with actual data)
    storage_capacity = {1: 580, 2: 687, 3: 599, 4: 788, 5: 294}
    production_capacity = {1: 1080, 2: 908, 3: 408, 4: 1000, 5: 403}

    # Create a Pyomo model
    model = pyo.ConcreteModel()

    # Define sets
    model.food_products = pyo.Set(initialize=[1, 2])
    model.beverage_products = pyo.Set(initialize=[1, 2, 3])
    model.time_periods = pyo.Set(initialize=list(range(1, 13)))

    # Define decision variables
    model.x = pyo.Var(model.food_products, model.time_periods, domain=pyo.NonNegativeIntegers)
    model.s = pyo.Var(model.food_products, model.time_periods, domain=pyo.NonNegativeIntegers)
    model.y = pyo.Var(model.food_products, model.time_periods, domain=pyo.Binary)
    model.x_bev = pyo.Var(model.beverage_products, model.time_periods, domain=pyo.NonNegativeReals)
    model.s_bev = pyo.Var(model.beverage_products, model.time_periods, domain=pyo.NonNegativeReals)
    model.y_bev = pyo.Var(model.beverage_products, model.time_periods, domain=pyo.Binary)

    # Define objective function
    def objective_rule(model):
        food_revenue = sum(revenue.loc[i, t] * min(model.s[i, t-1] + model.x[i, t], demand.loc[i, t])
                           for i in model.food_products for t in model.time_periods)
        food_cost = sum(fixed_cost_production.loc[i, t] * model.y[i, t] +
                        variable_cost_production.loc[i, t] * model.x[i, t] +
                        variable_cost_storage.loc[i, t] * model.s[i, t]
                        for i in model.food_products for t in model.time_periods)
        beverage_revenue = sum(revenue.loc[i+2, t] * min(model.s_bev[i, t-1] + model.x_bev[i, t], demand.loc[i+2, t])
                               for i in model.beverage_products for t in model.time_periods)
        beverage_cost = sum(fixed_cost_production.loc[i+2, t] * model.y_bev[i, t] +
                            variable_cost_production.loc[i+2, t] * model.x_bev[i, t] +
                            variable_cost_storage.loc[i+2, t] * model.s_bev[i, t]
                            for i in model.beverage_products for t in model.time_periods)
        return food_revenue - food_cost + beverage_revenue - beverage_cost

    model.objective = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

    # Define constraints
    def storage_balance_food_rule(model, i, t):
        if t == 1:
            return model.s[i, t] == model.x[i, t] - min(model.x[i, t], demand.loc[i, t])
        else:
            return model.s[i, t] == model.s[i, t-1] + model.x[i, t] - min(model.s[i, t-1] + model.x[i, t], demand.loc[i, t])

    model.storage_balance_food = pyo.Constraint(model.food_products, model.time_periods, rule=storage_balance_food_rule)

    def storage_balance_beverage_rule(model, i, t):
        if t == 1:
            return model.s_bev[i, t] == model.x_bev[i, t] - min(model.x_bev[i, t], demand.loc[i+2, t])
        else:
            return model.s_bev[i, t] == model.s_bev[i, t-1] + model.x_bev[i, t] - min(model.s_bev[i, t-1] + model.x_bev[i, t], de

    model.storage_balance_beverage = pyo.Constraint(model.beverage_products, model.time_periods, rule=storage_balance_beverage_ru

    def storage_capacity_food_rule(model, i, t):
        return model.s[i, t] <= storage_capacity[i]

    model.storage_capacity_food = pyo.Constraint(model.food_products, model.time_periods, rule=storage_capacity_food_rule)

    def storage_capacity_beverage_rule(model, i, t):
        return model.s_bev[i, t] <= storage_capacity[i+2]

    model.storage_capacity_beverage = pyo.Constraint(model.beverage_products, model.time_periods, rule=storage_capacity_beverage_

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

```
    return model.x[i, t] <= production_capacity[i] * model.y[i, t]

model.production_capacity_food = pyo.Constraint(model.food_products, model.time_periods, rule=production_capacity_food_rule)

def production_capacity_beverage_rule(model, i, t):
    return model.x_bev[i, t] <= production_capacity[i+2] * model.y_bev[i, t]

model.production_capacity_beverage = pyo.Constraint(model.beverage_products, model.time_periods, rule=production_capacity_bev

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

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

print("Food products:")
for i in model.food_products:
    for t in model.time_periods:
        print(f"Product {i}, Period {t}: Production = {model.x[i, t].value}, Storage = {model.s[i, t].value}")

print("Beverage products:")
for i in model.beverage_products:
    for t in model.time_periods:
        print(f"Product {i}, Period {t}: Production = {model.x_bev[i, t].value}, Storage = {model.s_bev[i, t].value}")
```

```
ERROR:pyomo.core:Rule failed when generating expression for Objective objective with index None:
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-15-441070e6784a> in <cell line: 66>()
     64     return food_revenue - food_cost + beverage_revenue - beverage_cost
     65 
---> 66 model.objective = pyo.Objective(rule=objective_rule, sense=pyo.maximize)
     67 
     68 # 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)

KeyError: "Index '(1, 0)' is not valid for indexed component 's'"
```

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
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)

# Sample data for storage and production capacities (replace with actual data)
storage_capacity = {1: 580, 2: 687, 3: 599, 4: 788, 5: 294}
production_capacity = {1: 1080, 2: 908, 3: 408, 4: 1000, 5: 403}

# Create a Pyomo model
model = pyo.ConcreteModel()

# Define sets
model.food_products = pyo.Set(initialize=[1, 2])
model.beverage_products = pyo.Set(initialize=[1, 2, 3])
model.time_periods = pyo.Set(initialize=list(range(1, 13)))

# Define decision variables
model.x = pyo.Var(model.food_products, model.time_periods, domain=pyo.NonNegativeIntegers)
model.s = pyo.Var(model.food_products, model.time_periods, domain=pyo.NonNegativeIntegers)
model.y = pyo.Var(model.food_products, model.time_periods, domain=pyo.Binary)
model.m = pyo.Var(model.food_products, model.time_periods, domain=pyo.NonNegativeIntegers)
model.x_bev = pyo.Var(model.beverage_products, model.time_periods, domain=pyo.NonNegativeReals)
model.s_bev = pyo.Var(model.beverage_products, model.time_periods, domain=pyo.NonNegativeReals)
model.y_bev = pyo.Var(model.beverage_products, model.time_periods, domain=pyo.Binary)
model.m_bev = pyo.Var(model.beverage_products, model.time_periods, domain=pyo.NonNegativeReals)

# Define objective function
def objective_rule(model):
    food_revenue = sum(revenue.loc[i, t] * model.m[i, t]
                       for i in model.food_products for t in model.time_periods)
    food_cost = sum(fixed_cost_production.loc[i, t] * model.y[i, t] +
                    variable_cost_production.loc[i, t] * model.x[i, t] +
                    variable_cost_storage.loc[i, t] * model.s[i, t]
                    for i in model.food_products for t in model.time_periods)
    beverage_revenue = sum(revenue.loc[i+2, t] * model.m_bev[i, t]
                           for i in model.beverage_products for t in model.time_periods)
    beverage_cost = sum(fixed_cost_production.loc[i+2, t] * model.y_bev[i, t] +
                        variable_cost_production.loc[i+2, t] * model.x_bev[i, t] +
                        variable_cost_storage.loc[i+2, t] * model.s_bev[i, t]
                        for i in model.beverage_products for t in model.time_periods)
    return food_revenue - food_cost + beverage_revenue - beverage_cost

model.objective = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

# Define constraints
def storage_balance_food_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_food = pyo.Constraint(model.food_products, model.time_periods, rule=storage_balance_food_rule)

def storage_balance_beverage_rule(model, i, t):
    if t == 1:
        return model.s_bev[i, t] == model.x_bev[i, t] - model.m_bev[i, t]
    else:
        return model.s_bev[i, t] == model.s_bev[i, t-1] + model.x_bev[i, t] - model.m_bev[i, t]

model.storage_balance_beverage = pyo.Constraint(model.beverage_products, model.time_periods, rule=storage_balance_beverage_ru

def storage_capacity_food_rule(model, i, t):
    return model.s[i, t] <= storage_capacity[i]

model.storage_capacity_food = pyo.Constraint(model.food_products, model.time_periods, rule=storage_capacity_food_rule)

def storage_capacity_beverage_rule(model, i, t):
    return model.s_bev[i, t] <= storage_capacity[i+2]

model.storage_capacity_beverage = pyo.Constraint(model.beverage_products, model.time_periods, rule=storage_capacity_beverage_

def production_capacity_food_rule(model, i, t):
    return model.x[i, t] <= production_capacity[i] * model.y[i, t]

model.production_capacity_food = pyo.Constraint(model.food_products, model.time_periods, rule=production_capacity_food_rule)

def production_capacity_beverage_rule(model, i, t):
    return model.x_bev[i, t] <= production_capacity[i+2] * model.y_bev[i, t]

model.production_capacity_beverage = pyo.Constraint(model.beverage_products, model.time_periods, rule=production_capacity_bev

# Auxiliary constraints
def min_demand_constraint_food_rule(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_food_rule = pyo.Constraint(model.food_products, model.time_periods, rule=min_demand_constraint_fo

def min_demand_constraint_bev_rule(model, i, t):
    if t == 1:
        return model.m_bev[i, t] <= model.x_bev[i, t] + 0
    else:
        return model.m_bev[i, t] <= model.x_bev[i, t] + model.s_bev[i, t-1]
model.min_demand_constraint_bev_rule = pyo.Constraint(model.beverage_products, model.time_periods, rule=min_demand_constraint_

def min_demand_constraint_demand_food_rule(model, i, t):
    return model.m[i, t] <= demand[t][i]
model.min_demand_constraint_demand_food_rule = pyo.Constraint(model.food_products, model.time_periods, rule=min_demand_constr

def min_demand_constraint_demand_bev_rule(model, i, t):
    return model.m_bev[i, t] <= demand[t][i+2]
model.min_demand_constraint_demand_bev_rule = pyo.Constraint(model.beverage_products, model.time_periods, rule=min_demand_con

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

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

print("Food products:")
for i in model.food_products:
    for t in model.time_periods:
        print(f"Product {i}, Period {t}: Production = {model.x[i, t].value}, Storage = {model.s[i, t].value}")

print("Beverage products:")
for i in model.beverage_products:
    for t in model.time_periods:
        print(f"Product {i}, Period {t}: Production = {model.x_bev[i, t].value}, Storage = {model.s_bev[i, t].value}")
```

```
Objective value: 4523.2892999999995
Food products:
Product 1, Period 1: Production = -0.0, Storage = -0.0
Product 1, Period 2: Production = -0.0, Storage = -0.0
Product 1, Period 3: Production = -0.0, Storage = -0.0
Product 1, Period 4: Production = -0.0, Storage = -0.0
Product 1, Period 5: Production = -0.0, Storage = -0.0
Product 1, Period 6: Production = -0.0, Storage = -0.0
Product 1, Period 7: Production = -0.0, Storage = -0.0
Product 1, Period 8: Production = -0.0, Storage = -0.0
Product 1, Period 9: Production = -0.0, Storage = -0.0
Product 1, Period 10: Production = -0.0, Storage = -0.0
Product 1, Period 11: Production = -0.0, Storage = -0.0
Product 1, Period 12: Production = -0.0, Storage = -0.0
Product 2, Period 1: Production = -0.0, Storage = -0.0
Product 2, Period 2: Production = -0.0, Storage = -0.0
Product 2, Period 3: Production = -0.0, Storage = -0.0
Product 2, Period 4: Production = -0.0, Storage = -0.0
Product 2, Period 5: Production = -0.0, Storage = -0.0
Product 2, Period 6: Production = -0.0, Storage = -0.0
Product 2, Period 7: Production = -0.0, Storage = -0.0
Product 2, Period 8: Production = -0.0, Storage = -0.0
Product 2, Period 9: Production = -0.0, Storage = -0.0
Product 2, Period 10: Production = -0.0, Storage = -0.0
Product 2, Period 11: Production = -0.0, Storage = -0.0
Product 2, Period 12: Production = -0.0, Storage = -0.0
Beverage products:
Product 1, Period 1: Production = 13.879999999999999, Storage = 8.2
Product 1, Period 2: Production = 0.0, Storage = 0.0
Product 1, Period 3: Production = 13.68, Storage = 5.83
Product 1, Period 4: Production = 0.0, Storage = 0.0
Product 1, Period 5: Production = 14.18, Storage = 6.13
Product 1, Period 6: Production = 0.0, Storage = 0.0
Product 1, Period 7: Production = 20.43, Storage = 14.05
Product 1, Period 8: Production = 0.0, Storage = 6.75
Product 1, Period 9: Production = 0.0, Storage = 0.0
Product 1, Period 10: Production = 11.690000000000001, Storage = 5.86
Product 1, Period 11: Production = 0.0, Storage = 0.0
Product 1, Period 12: Production = 5.85, Storage = 0.0
Product 2, Period 1: Production = 0.0, Storage = 0.0
Product 2, Period 2: Production = 0.0, Storage = 0.0
Product 2, Period 3: Production = 0.0, Storage = 0.0
Product 2, Period 4: Production = 0.0, Storage = 0.0
Product 2, Period 5: Production = 0.0, Storage = 0.0
Product 2, Period 6: Production = 0.0, Storage = 0.0
Product 2, Period 7: Production = 0.0, Storage = 0.0
Product 2, Period 8: Production = 0.0, Storage = 0.0
Product 2, Period 9: Production = 0.0, Storage = 0.0
Product 2, Period 10: Production = 0.0, Storage = 0.0
Product 2, Period 11: Production = 0.0, Storage = 0.0
Product 2, Period 12: Production = 0.0, Storage = 0.0
Product 3, Period 1: Production = 0.0, Storage = 0.0
Product 3, Period 2: Production = 0.0, Storage = 0.0
Product 3, Period 3: Production = 0.0, Storage = 0.0
Product 3, Period 4: Production = 0.0, Storage = 0.0
Product 3, Period 5: Production = 0.0, Storage = 0.0
Product 3, Period 6: Production = 7.79, Storage = 3.94
Product 3, Period 7: Production = 0.0, Storage = 0.0
```