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
prompt1_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt1_MathematicalModel.txt'
prompt2_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt2_PyomoCode.txt'
problem_desc_path = '/content/drive/MyDrive/Thesis/ProblemDescriptions/MIP/MIP2.txt'

prompt1_file = open(prompt1_path, "r")
prompt2_file = open(prompt2_path, "r")
problem_desc_file = open(problem_desc_path, "r")

prompt1 = prompt1_file.read()
print("Prompt 1:\n", prompt1)

prompt2 = prompt2_file.read()
print("Prompt 2:\n", prompt2)

problem_desc = problem_desc_file.read()
print("Problem Description:\n", problem_desc)
```

```
Prompt 1:
 Please write a mathematical optimization model for this problem. Include parameters, decision variables, the objective
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 fo
```

```
!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: Create Mathematical Model

```
message = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt1,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)
```

```
response_p1 = message.content[0].text


# Print response
print(response_p1)
```

```
Thank you for providing the detailed problem description. Here is a mathematical optimization model for the food and bev

Parameters:
- T: Set of time periods (1, 2, ..., 12)
- P: Set of products (1, 2, ..., 5), where products 1 and 2 are food, and products 3, 4, and 5 are beverages
- FC[p, t]: Fixed cost of producing product p in period t (from "fixed_cost_production.csv")
- VCP[p, t]: Variable cost of producing product p in period t (from "variable_cost_production.csv")
- VCS[p, t]: Variable cost of storing product p in period t (from "variable_cost_storage.csv")
- D[p, t]: Maximum demand for product p in period t (from "demand.csv")
- R[p, t]: Revenue per unit of product p sold in period t (from "revenue.csv")
- SC[p]: Storage capacity for product p (580 for food 1, 687 for food 2, 599 for beverage 1, 788 for beverage 2, 294 for
- PC[p]: Production capacity for product p (1080 for food 1, 908 for food 2, 408 for beverage 1, 1000 for beverage 2, 40

Decision Variables:
- X[p, t]: Quantity of product p produced in period t (integer for food, continuous for beverages)
- S[p, t]: Quantity of product p stored at the end of period t (integer for food, continuous for beverages)
- Y[p, t]: Binary variable indicating whether product p is produced in period t (1 if produced, 0 otherwise)

Objective Function:
Maximize: Sum[(p, t) in P × T] (R[p, t] × (X[p, t] - S[p, t]) - FC[p, t] × Y[p, t] - VCP[p, t] × X[p, t] - VCS[p, t] × S

Constraints:
1. Inventory Balance: For each product p and period t,
   S[p, t] = S[p, t-1] + X[p, t] - (X[p, t] - S[p, t]), where S[p, 0] = 0

2. Demand Limit: For each product p and period t,
   X[p, t] - S[p, t] ≤ D[p, t]

3. Storage Capacity: For each product p and period t,
   S[p, t] ≤ SC[p]

4. Production Capacity: For each product p and period t,
   X[p, t] ≤ PC[p] × Y[p, t]

5. Fixed Cost Indicator: For each product p and period t,
   X[p, t] ≤ M × Y[p, t], where M is a large positive number

6. Food Production Integrality: For products 1 and 2 (food) and each period t,
   X[p, t], S[p, t] are non-negative integers

7. Beverage Production Non-negativity: For products 3, 4, and 5 (beverages) and each period t,
   X[p, t], S[p, t] ≥ 0

The objective function aims to maximize the total profit, which is calculated as the revenue from sold products minus th
```

2. Prompt 2: Write Code based on Mathematical Model

```
message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": response_p1}
    ]
)

response_p2 = message2.content[0].text


# Print response
print(response_p2)
```

```
Here's the Python Pyomo code for the food and beverage production planning optimization problem:

```python
import pyomo.environ as pyo
import pandas as pd

# Read input data from CSV files
fixed_cost_production = pd.read_csv("fixed_cost_production.csv", index_col=[0, 1])
variable_cost_production = pd.read_csv("variable_cost_production.csv", index_col=[0, 1])
variable_cost_storage = pd.read_csv("variable_cost_storage.csv", index_col=[0, 1])
demand = pd.read_csv("demand.csv", index_col=[0, 1])
revenue = pd.read_csv("revenue.csv", index_col=[0, 1])

# Create the model
model = pyo.ConcreteModel()
```
```

```
    # Define sets
    model.T = pyo.Set(initialize=range(1, 13))  # Time periods
    model.P = pyo.Set(initialize=range(1, 6))   # Products

    # Define parameters
    model.FC = pyo.Param(model.P, model.T, initialize=fixed_cost_production.stack().to_dict())
    model.VCP = pyo.Param(model.P, model.T, initialize=variable_cost_production.stack().to_dict())
    model.VCS = pyo.Param(model.P, model.T, initialize=variable_cost_storage.stack().to_dict())
    model.D = pyo.Param(model.P, model.T, initialize=demand.stack().to_dict())
    model.R = pyo.Param(model.P, model.T, initialize=revenue.stack().to_dict())
    model.SC = pyo.Param(model.P, initialize={1: 580, 2: 687, 3: 599, 4: 788, 5: 294})  # Sample data
    model.PC = pyo.Param(model.P, initialize={1: 1080, 2: 908, 3: 408, 4: 1000, 5: 403})  # Sample data

    # Define decision variables
    model.X = pyo.Var(model.P, model.T, domain=pyo.NonNegativeReals)
    model.S = pyo.Var(model.P, model.T, domain=pyo.NonNegativeReals)
    model.Y = pyo.Var(model.P, model.T, domain=pyo.Binary)

    # Define objective function
    def objective_rule(model):
        return pyo.summation(model.R, model.X - model.S) - pyo.summation(model.FC, model.Y) - \
               pyo.summation(model.VCP, model.X) - pyo.summation(model.VCS, model.S)
    model.OBJ = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

    # Define constraints
    def inventory_balance_rule(model, p, t):
        if t == 1:
            return model.S[p, t] == model.X[p, t] - (model.X[p, t] - model.S[p, t])
        else:
            return model.S[p, t] == model.S[p, t-1] + model.X[p, t] - (model.X[p, t] - model.S[p, t])
    model.InventoryBalance = pyo.Constraint(model.P, model.T, rule=inventory_balance_rule)

    def demand_limit_rule(model, p, t):
        return model.X[p, t] - model.S[p, t] <= model.D[p, t]
    model.DemandLimit = pyo.Constraint(model.P, model.T, rule=demand_limit_rule)

    def storage_capacity_rule(model, p, t):
        return model.S[p, t] <= model.SC[p]
    model.StorageCapacity = pyo.Constraint(model.P, model.T, rule=storage_capacity_rule)

    def production_capacity_rule(model, p, t):
```

4. Input Problem Data and Test Model Code

```
# Installing pyomo nd solver
!pip install -q pyomo
!apt-get install -y -qq glpk-utils
!pip install glpk
!pip install pandas
```

Show hidden output

```
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.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)

# Create the model
model = pyo.ConcreteModel()

# Define sets
model.T = pyo.Set(initialize=range(1, 13))  # Time periods
model.P = pyo.Set(initialize=range(1, 6))   # Products

# Define parameters
model.FC = pyo.Param(model.P, model.T, initialize=fixed_cost_production.stack().to_dict())
model.VCP = pyo.Param(model.P, model.T, initialize=variable_cost_production.stack().to_dict())
model.VCS = pyo.Param(model.P, model.T, initialize=variable_cost_storage.stack().to_dict())
model.D = pyo.Param(model.P, model.T, initialize=demand.stack().to_dict())
model.R = pyo.Param(model.P, model.T, initialize=revenue.stack().to_dict())
model.SC = pyo.Param(model.P, initialize={1: 580, 2: 687, 3: 599, 4: 788, 5: 294})  # Sample data
model.PC = pyo.Param(model.P, initialize={1: 1080, 2: 908, 3: 408, 4: 1000, 5: 403})  # Sample data

# Define decision variables
model.X = pyo.Var(model.P, model.T, domain=pyo.NonNegativeReals)
model.S = pyo.Var(model.P, model.T, domain=pyo.NonNegativeReals)
model.Y = pyo.Var(model.P, model.T, domain=pyo.Binary)

# Define objective function
def objective_rule(model):
    return pyo.summation(model.R, model.X - model.S) - pyo.summation(model.FC, model.Y) - \
           pyo.summation(model.VCP, model.X) - pyo.summation(model.VCS, model.S)
model.OBJ = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

# Define constraints
def inventory_balance_rule(model, p, t):
    if t == 1:
        return model.S[p, t] == model.X[p, t] - (model.X[p, t] - model.S[p, t])
    else:
        return model.S[p, t] == model.S[p, t-1] + model.X[p, t] - (model.X[p, t] - model.S[p, t])
model.InventoryBalance = pyo.Constraint(model.P, model.T, rule=inventory_balance_rule)

def demand_limit_rule(model, p, t):
    return model.X[p, t] - model.S[p, t] <= model.D[p, t]
model.DemandLimit = pyo.Constraint(model.P, model.T, rule=demand_limit_rule)

def storage_capacity_rule(model, p, t):
    return model.S[p, t] <= model.SC[p]
model.StorageCapacity = pyo.Constraint(model.P, model.T, rule=storage_capacity_rule)

def production_capacity_rule(model, p, t):
    return model.X[p, t] <= model.PC[p] * model.Y[p, t]
model.ProductionCapacity = pyo.Constraint(model.P, model.T, rule=production_capacity_rule)

def fixed_cost_indicator_rule(model, p, t):
    return model.X[p, t] <= 1e6 * model.Y[p, t]  # Using 1e6 as a large positive number
model.FixedCostIndicator = pyo.Constraint(model.P, model.T, rule=fixed_cost_indicator_rule)

def food_production_integrality_rule(model, p, t):
    if p in [1, 2]:
```

```
        return model.X[p, t].within(pyo.Integers)
    else:
        return pyo.Constraint.Skip
model.FoodProductionIntegrality = pyo.Constraint(model.P, model.T, rule=food_production_integrality_rule)

def food_storage_integrality_rule(model, p, t):
    if p in [1, 2]:
        return model.S[p, t].within(pyo.Integers)
    else:
        return pyo.Constraint.Skip
model.FoodStorageIntegrality = pyo.Constraint(model.P, model.T, rule=food_storage_integrality_rule)

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

# Print the results
print(results)

for p in model.P:
    for t in model.T:
        print(f"Product {p} in Period {t}:")
        print(f"  Production: {model.X[p, t].value}")
        print(f"  Storage: {model.S[p, t].value}")
        print(f"  Production Binary: {model.Y[p, t].value}")
```

```
ERROR:pyomo.core:Rule failed when generating expression for Objective OBJ with index None:
TypeError: unsupported operand type(s) for -: 'IndexedVar' and 'IndexedVar'
ERROR:pyomo.core:Constructing component 'OBJ' from data=None failed:
    TypeError: unsupported operand type(s) for -: 'IndexedVar' and 'IndexedVar'
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-ebe9f7e47e6d> in <cell line: 55>()
     53     return pyo.summation(model.R, model.X - model.S) - pyo.summation(model.FC, model.Y) - \
     54            pyo.summation(model.VCP, model.X) - pyo.summation(model.VCS, model.S)
---> 55 model.OBJ = pyo.Objective(rule=objective_rule, sense=pyo.maximize)
     56 
     57 # Define constraints

                                    4 frames
<ipython-input-4-ebe9f7e47e6d> in objective_rule(model)
     51 # Define objective function
     52 def objective_rule(model):
---> 53     return pyo.summation(model.R, model.X - model.S) - pyo.summation(model.FC, model.Y) - \
     54            pyo.summation(model.VCP, model.X) - pyo.summation(model.VCS, model.S)
     55 model.OBJ = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

TypeError: unsupported operand type(s) for -: 'IndexedVar' and 'IndexedVar'
```

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.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)

# Create the model
model = pyo.ConcreteModel()

# Define sets
model.T = pyo.Set(initialize=range(1, 13))  # Time periods
model.P = pyo.Set(initialize=range(1, 6))   # Products

# Define parameters
model.FC = pyo.Param(model.P, model.T, initialize=fixed_cost_production.stack().to_dict())
model.VCP = pyo.Param(model.P, model.T, initialize=variable_cost_production.stack().to_dict())
model.VCS = pyo.Param(model.P, model.T, initialize=variable_cost_storage.stack().to_dict())
model.D = pyo.Param(model.P, model.T, initialize=demand.stack().to_dict())
model.R = pyo.Param(model.P, model.T, initialize=revenue.stack().to_dict())
model.SC = pyo.Param(model.P, initialize={1: 580, 2: 687, 3: 599, 4: 788, 5: 294})  # Sample data
model.PC = pyo.Param(model.P, initialize={1: 1080, 2: 908, 3: 408, 4: 1000, 5: 403})  # Sample data

# Define decision variables
model.X = pyo.Var(model.P, model.T, domain=pyo.NonNegativeReals)
model.S = pyo.Var(model.P, model.T, domain=pyo.NonNegativeReals)
model.Y = pyo.Var(model.P, model.T, domain=pyo.Binary)

#set domain to integer for food MODIFIED ADDED INTEGRALITY HERE INSTEAD OF CONSTRAINT
for t in model.T:
    model.X[2, t].domain = pyo.NonNegativeIntegers
    model.X[1, t].domain = pyo.NonNegativeIntegers
    model.S[2, t].domain = pyo.NonNegativeIntegers
    model.S[1, t].domain = pyo.NonNegativeIntegers

# Define objective function
def objective_rule(model):
    total_revenue = sum(model.R[p, t] * (model.X[p, t] - model.S[p, t]) for p in model.P for t in model.T)
    total_fixed_cost = sum(model.FC[p, t] * model.Y[p, t] for p in model.P for t in model.T)
    total_variable_production_cost = sum(model.VCP[p, t] * model.X[p, t] for p in model.P for t in model.T)
    total_variable_storage_cost = sum(model.VCS[p, t] * model.S[p, t] for p in model.P for t in model.T)
    return total_revenue - total_fixed_cost - total_variable_production_cost - total_variable_storage_cost
model.OBJ = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

# Define constraints
def inventory_balance_rule(model, p, t):
    if t == 1:
        return model.S[p, t] == model.X[p, t] - (model.X[p, t] - model.S[p, t])
    else:
        return model.S[p, t] == model.S[p, t-1] + model.X[p, t] - (model.X[p, t] - model.S[p, t])
model.InventoryBalance = pyo.Constraint(model.P, model.T, rule=inventory_balance_rule)

def demand_limit_rule(model, p, t):
    return model.X[p, t] - model.S[p, t] <= model.D[p, t]
model.DemandLimit = pyo.Constraint(model.P, model.T, rule=demand_limit_rule)

def storage_capacity_rule(model, p, t):
    return model.S[p, t] <= model.SC[p]
model.StorageCapacity = pyo.Constraint(model.P, model.T, rule=storage_capacity_rule)

def production_capacity_rule(model, p, t):
    return model.X[p, t] <= model.PC[p] * model.Y[p, t]
model.ProductionCapacity = pyo.Constraint(model.P, model.T, rule=production_capacity_rule)

def fixed_cost_indicator_rule(model, p, t):
    return model.X[p, t] <= 1e6 * model.Y[p, t]  # Using 1e6 as a large positive number
model.FixedCostIndicator = pyo.Constraint(model.P, model.T, rule=fixed_cost_indicator_rule)

# Solve the model
```

```
    solver = pyo.SolverFactory('gurobi')
    results = solver.solve(model)

    # Print the results
    print(results)

    for p in model.P:
        for t in model.T:
            print(f"Product {p} in Period {t}:")
            print(f"  Production: {model.X[p, t].value}")
            print(f"  Storage: {model.S[p, t].value}")
            print(f"  Production Binary: {model.Y[p, t].value}")
```

```
  Production: 0.0
  Storage: -0.0
  Production Binary: 0.0
Product 4 in Period 6:
  Production: 0.0
  Storage: -0.0
  Production Binary: 0.0
Product 4 in Period 7:
  Production: 0.0
  Storage: -0.0
  Production Binary: 0.0
Product 4 in Period 8:
  Production: 0.0
  Storage: -0.0
  Production Binary: 0.0
Product 4 in Period 9:
  Production: 0.0
  Storage: -0.0
  Production Binary: 0.0
Product 4 in Period 10:
  Production: 0.0
  Storage: -0.0
  Production Binary: 0.0
Product 4 in Period 11:
  Production: 0.0
  Storage: -0.0
  Production Binary: 0.0
Product 4 in Period 12:
  Production: 0.0
  Storage: 0.0
  Production Binary: 0.0
Product 5 in Period 1:
  Production: 0.0
  Storage: -0.0
  Production Binary: 0.0
Product 5 in Period 2:
  Production: 0.0
  Storage: -0.0
  Production Binary: 0.0
Product 5 in Period 3:
  Production: 0.0
  Storage: -0.0
  Production Binary: 0.0
Product 5 in Period 4:
  Production: 0.0
  Storage: -0.0
  Production Binary: 0.0
Product 5 in Period 5:
  Production: 0.0
```