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 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: 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)
```

```
To solve this production planning problem and maximize profit, we can formulate a mathematical optimization model. Here

    Parameters:
    - T: Set of time periods (1 to 12)
    - P: Set of products (food1, food2, beverage1, beverage2, beverage3)
    - fixed_cost[p,t]: Fixed cost of producing product p in period t
    - variable_cost_production[p,t]: Variable cost of producing one unit of product p in period t
    - variable_cost_storage[p,t]: Variable cost of storing one unit of product p in period t
    - demand[p,t]: Maximum demand for product p in period t
    - revenue[p,t]: Revenue per unit of product p sold in period t
    - storage_capacity[p]: Storage capacity for product p
    - production_capacity[p]: Production capacity for product p

    Decision Variables:
    - production[p,t]: Quantity of product p produced in period t (integer for food, continuous for beverages)
    - storage[p,t]: Quantity of product p stored at the end of period t (integer for food, continuous for beverages)
    - sales[p,t]: Quantity of product p sold in period t (integer for food, continuous for beverages)

    Objective Function:
    Maximize: sum(sales[p,t] * revenue[p,t] - production[p,t] * (fixed_cost[p,t] + variable_cost_production[p,t]) - storage[

    Constraints:
    1. Production Capacity:
       - production[p,t] <= production_capacity[p] for all p in P and t in T

    2. Storage Capacity:
       - storage[p,t] <= storage_capacity[p] for all p in P and t in T

    3. Demand Satisfaction:
       - sales[p,t] <= demand[p,t] for all p in P and t in T

    4. Inventory Balance:
       - storage[p,t] = storage[p,t-1] + production[p,t] - sales[p,t] for all p in P and t in T (t > 1)
       - storage[p,1] = production[p,1] - sales[p,1] for all p in P (initial period)

    5. Non-negativity:
       - production[p,t], storage[p,t], sales[p,t] >= 0 for all p in P and t in T

    6. Integer Constraints:
       - production[p,t], storage[p,t], sales[p,t] are integers for food products (food1 and food2)

    The objective function aims to maximize the total profit by considering the revenue from sales, fixed and variable produ

    To solve this model, you would need to input the data from the provided CSV files into the corresponding parameters and
```

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)
```

```
model.storage_capacity_constraint = Constraint(model.P, model.T, rule=storage_capacity_rule)

def demand_satisfaction_rule(model, p, t):
    return model.sales[p,t] <= model.demand[p,t]

model.demand_satisfaction_constraint = Constraint(model.P, model.T, rule=demand_satisfaction_rule)

def inventory_balance_rule(model, p, t):
    if t == 1:
        return model.storage[p,t] == model.production[p,t] - model.sales[p,t]
    else:
        return model.storage[p,t] == model.storage[p,t-1] + model.production[p,t] - model.sales[p,t]

model.inventory_balance_constraint = Constraint(model.P, model.T, rule=inventory_balance_rule)

# Define integer constraints for food products
def integer_constraint_rule(model, p, t):
    if p in ['food1', 'food2']:
        return model.production[p,t].is_integer() and model.storage[p,t].is_integer() and model.sales[p,t].is_integer(
    else:
        return Constraint.Skip

model.integer_constraint = Constraint(model.P, model.T, rule=integer_constraint_rule)

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

# Print the results
print("Production Plan:")
for p in model.P:
    for t in model.T:
        print(f"Product: {p}, Period: {t}, Production: {model.production[p,t].value}")

print("\nStorage Plan:")
for p in model.P:
    for t in model.T:
        print(f"Product: {p}, Period: {t}, Storage: {model.storage[p,t].value}")

print("\nSales Plan:")
for p in model.P:
    for t in model.T:
        print(f"Product: {p}, Period: {t}, Sales: {model.sales[p,t].value}")

print(f"\nMaximum Profit: {model.profit()}")
```

In this code, we assume that the input data is provided in CSV files named 'demand.csv', 'fixed_cost.csv', 'variable_c

For storage and production capacities, sample data is provided directly in the code using dictionaries (`storage_capac

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 pandas as pd
from pyomo.environ import *

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

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

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

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

```
demand_data.columns = demand_data.columns.astype(int)

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

# Create the Pyomo model
model = ConcreteModel()

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

# Define parameters
model.fixed_cost = Param(model.P, model.T, initialize=fixed_cost_data.stack().to_dict())
model.variable_cost_production = Param(model.P, model.T, initialize=variable_cost_production_data.stack().to_dict())
model.variable_cost_storage = Param(model.P, model.T, initialize=variable_cost_storage_data.stack().to_dict())
model.demand = Param(model.P, model.T, initialize=demand_data.stack().to_dict())
model.revenue = Param(model.P, model.T, initialize=revenue_data.stack().to_dict())

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

model.storage_capacity = Param(model.P, initialize=storage_capacity)
model.production_capacity = Param(model.P, initialize=production_capacity)

# Define decision variables
model.production = Var(model.P, model.T, domain=NonNegativeReals)
model.storage = Var(model.P, model.T, domain=NonNegativeReals)
model.sales = Var(model.P, model.T, domain=NonNegativeReals)

# Define objective function
def profit_rule(model):
    profit = sum(model.sales[p,t] * model.revenue[p,t] - model.production[p,t] * (model.fixed_cost[p,t] + model.variable_cost
    return profit

model.profit = Objective(rule=profit_rule, sense=maximize)

# Define constraints
def production_capacity_rule(model, p, t):
    return model.production[p,t] <= model.production_capacity[p]

model.production_capacity_constraint = Constraint(model.P, model.T, rule=production_capacity_rule)

def storage_capacity_rule(model, p, t):
    return model.storage[p,t] <= model.storage_capacity[p]

model.storage_capacity_constraint = Constraint(model.P, model.T, rule=storage_capacity_rule)

def demand_satisfaction_rule(model, p, t):
    return model.sales[p,t] <= model.demand[p,t]

model.demand_satisfaction_constraint = Constraint(model.P, model.T, rule=demand_satisfaction_rule)

def inventory_balance_rule(model, p, t):
    if t == 1:
        return model.storage[p,t] == model.production[p,t] - model.sales[p,t]
    else:
        return model.storage[p,t] == model.storage[p,t-1] + model.production[p,t] - model.sales[p,t]

model.inventory_balance_constraint = Constraint(model.P, model.T, rule=inventory_balance_rule)

# Define integer constraints for food products
def integer_constraint_rule(model, p, t):
    if p in [1, 2]:
        return model.production[p,t].is_integer() and model.storage[p,t].is_integer() and model.sales[p,t].is_integer()
    else:
        return Constraint.Skip

model.integer_constraint = Constraint(model.P, model.T, rule=integer_constraint_rule)
```

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

# Print the results
print("Production Plan:")
for p in model.P:
    for t in model.T:
        print(f"Product: {p}, Period: {t}, Production: {model.production[p,t].value}")

print("\nStorage Plan:")
for p in model.P:
    for t in model.T:
        print(f"Product: {p}, Period: {t}, Storage: {model.storage[p,t].value}")

print("\nSales Plan:")
for p in model.P:
    for t in model.T:
        print(f"Product: {p}, Period: {t}, Sales: {model.sales[p,t].value}")

print(f"\nMaximum Profit: {model.profit()}")
```

```
ERROR:pyomo.core:Rule failed when generating expression for Constraint intege
ValueError: Invalid constraint expression. The constraint expression resolved

Error thrown for Constraint 'integer_constraint[1,1]'
ERROR:pyomo.core:Constructing component 'integer_constraint' from data=None fa
    ValueError: Invalid constraint expression. The constraint expression reso

Error thrown for Constraint 'integer_constraint[1,1]'
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-14-36b259b5cabc> in <cell line: 104>()
    102         return Constraint.Skip
    103
--> 104 model.integer_constraint = Constraint(model.P, model.T,
rule=integer_constraint_rule)
    105
    106 # Solve the model

 ---------------------------------- 4 frames ----------------------------------
/usr/local/lib/python3.10/dist-packages/pyomo/core/base/constraint.py in
set_value(self, expr)
    399
    400         elif expr.__class__ is bool:
--> 401             raise ValueError(
    402                 "Invalid constraint expression. The constraint "
    403                 "expression resolved to a trivial Boolean (%s) "

ValueError: Invalid constraint expression. The constraint expression resolved
to a trivial Boolean (False) instead of a Pyomo object. Please modify your
```

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 pandas as pd
from pyomo.environ import *

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

```
fixed_cost_data.columns = fixed_cost_data.columns.astype(int)

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

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

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

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

# Create the Pyomo model
model = ConcreteModel()

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

# Define parameters
model.fixed_cost = Param(model.P, model.T, initialize=fixed_cost_data.stack().to_dict())
model.variable_cost_production = Param(model.P, model.T, initialize=variable_cost_production_data.stack().to_dict())
model.variable_cost_storage = Param(model.P, model.T, initialize=variable_cost_storage_data.stack().to_dict())
model.demand = Param(model.P, model.T, initialize=demand_data.stack().to_dict())
model.revenue = Param(model.P, model.T, initialize=revenue_data.stack().to_dict())

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

model.storage_capacity = Param(model.P, initialize=storage_capacity)
model.production_capacity = Param(model.P, initialize=production_capacity)

# Define decision variables
model.production = Var(model.P, model.T, domain=NonNegativeReals)
model.storage = Var(model.P, model.T, domain=NonNegativeReals)
model.sales = Var(model.P, model.T, domain=NonNegativeReals)

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

# Define objective function
def profit_rule(model):
    profit = sum(model.sales[p,t] * model.revenue[p,t] - model.production[p,t] * (model.fixed_cost[p,t] + model.variable_cost
    return profit

model.profit = Objective(rule=profit_rule, sense=maximize)

# Define constraints
def production_capacity_rule(model, p, t):
    return model.production[p,t] <= model.production_capacity[p]

model.production_capacity_constraint = Constraint(model.P, model.T, rule=production_capacity_rule)

def storage_capacity_rule(model, p, t):
    return model.storage[p,t] <= model.storage_capacity[p]
```

```
model.storage_capacity_constraint = Constraint(model.P, model.T, rule=storage_capacity_rule)

def demand_satisfaction_rule(model, p, t):
    return model.sales[p,t] <= model.demand[p,t]

model.demand_satisfaction_constraint = Constraint(model.P, model.T, rule=demand_satisfaction_rule)

def inventory_balance_rule(model, p, t):
    if t == 1:
        return model.storage[p,t] == model.production[p,t] - model.sales[p,t]
    else:
        return model.storage[p,t] == model.storage[p,t-1] + model.production[p,t] - model.sales[p,t]

model.inventory_balance_constraint = Constraint(model.P, model.T, rule=inventory_balance_rule)

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

# Print the results
print("Production Plan:")
for p in model.P:
    for t in model.T:
        print(f"Product: {p}, Period: {t}, Production: {model.production[p,t].value}")

print("\nStorage Plan:")
for p in model.P:
    for t in model.T:
        print(f"Product: {p}, Period: {t}, Storage: {model.storage[p,t].value}")

print("\nSales Plan:")
for p in model.P:
    for t in model.T:
        print(f"Product: {p}, Period: {t}, Sales: {model.sales[p,t].value}")

print(f"\nMaximum Profit: {model.profit()}")
```

```
Product: 4, Period: 5, Sales: 0.0
Product: 4, Period: 6, Sales: 0.0
Product: 4, Period: 7, Sales: 0.0
Product: 4, Period: 8, Sales: 0.0
Product: 4, Period: 9, Sales: 0.0
Product: 4, Period: 10, Sales: 0.0
Product: 4, Period: 11, Sales: 0.0
Product: 4, Period: 12, Sales: 0.0
Product: 5, Period: 1, Sales: 0.0
```