

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/LP/LP4.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:
 Consider a farmer who specializes in raising wheat, corn, and sugar beets on her 500 acres of land. During the winter, The farmer knows that at least 200 tons (T) of wheat and 240 T of corn are needed for cattle feed. These amounts can be Any production in excess of the feeding requirement would be sold.
 Over the last decade, mean selling prices have been \$170 and \$150 per ton of wheat and corn, respectively. The purchase prices are 40% more than this due to the wholesaler's margin and transportation costs.
 Another profitable crop is sugar beet, which she expects to sell at \$36/T. However, the European Commission imposes a qu Based on past experience, the farmer knows that the mean yield on her land is roughly 2.5 T, 3 T, and 20 T per acre for The farmer can decide how much of her land she will use to grow each of the three products: wheat, corn and sugar beets.

```
!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)
```

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

Decision Variables:

- Let W be the number of acres of land used to grow wheat
- Let C be the number of acres of land used to grow corn
- Let S be the number of acres of land used to grow sugar beets

Additional Variables:

- Let Wp be the number of tons of wheat purchased from the wholesaler
- Let Cp be the number of tons of corn purchased from the wholesaler
- Let Ws be the number of tons of wheat sold
- Let Cs be the number of tons of corn sold
- Let Sq be the number of tons of sugar beets sold at the quota price
- Let Se be the number of tons of sugar beets sold in excess of the quota


These variables will be used to formulate the objective function and constraints of the optimization problem, with the g

✓ 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 problem description and the defined variables, the objective function for the farmer's cost minimization pr

Minimize:

$$150W + 230C + 260S + 238Wp + 210Cp - 170Ws - 150Cs - 36Sq - 10Se$$

This objective function represents the total costs the farmer wants to minimize. Here's the breakdown:

- 150W, 230C, 260S: The planting costs for wheat, corn, and sugar beets, respectively, based on the number of acres allo
- 238Wp, 210Cp: The cost of purchasing wheat and corn from the wholesaler, calculated as the selling price plus a 40% ma
- 170Ws, 150Cs: The revenue generated from selling excess wheat and corn, treated as negative costs.
- 36Sq: The revenue from selling sugar beets within the quota, also treated as a negative cost.
- 10Se: The revenue from selling excess sugar beets beyond the quota, at the lower price of \$10/T.

The goal is to minimize this objective function, subject to the constraints that will be defined based on the land avail

✓ 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)
```

➦ Based on the given information and the defined variables, here are the constraints for the farmer's cost minimization pr

1. Land constraint:
 $W + C + S \leq 500$
 (The total acres allocated to wheat, corn, and sugar beets cannot exceed the available 500 acres)
2. Wheat production and purchase constraint:
 $2.5W + W_p \geq 200$
 (The sum of wheat produced on the farm and purchased from the wholesaler must meet the minimum requirement of 200 ton)
3. Corn production and purchase constraint:
 $3C + C_p \geq 240$
 (The sum of corn produced on the farm and purchased from the wholesaler must meet the minimum requirement of 240 tons)
4. Wheat balance constraint:
 $2.5W = 200 + W_s$
 (The wheat produced on the farm must equal the required 200 tons plus any excess sold)
5. Corn balance constraint:
 $3C = 240 + C_s$
 (The corn produced on the farm must equal the required 240 tons plus any excess sold)
6. Sugar beet quota constraint:
 $S_q \leq 6000$
 (The amount of sugar beets sold at the quota price cannot exceed the quota of 6000 tons)
7. Sugar beet balance constraint:
 $20S = S_q + S_e$
 (The total sugar beets produced on the farm must equal the sum of the amount sold at the quota price and the excess s
8. Non-negativity constraints:
 $W, C, S, W_p, C_p, W_s, C_s, S_q, S_e \geq 0$
 (All decision variables must be non-negative)

These constraints ensure that the farmer's land allocation, production, purchasing, and sales decisions are feasible and

✓ 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)

```

Here's the Python code using Pyomo to solve the farmer's cost minimization problem:

```

```python
from pyomo.environ import *

Create a Concrete Model
model = ConcreteModel()

Define decision variables
model.W = Var(domain=NonNegativeReals) # Acres of land for wheat
model.C = Var(domain=NonNegativeReals) # Acres of land for corn
model.S = Var(domain=NonNegativeReals) # Acres of land for sugar beets
model.Wp = Var(domain=NonNegativeReals) # Tons of wheat purchased
model.Cp = Var(domain=NonNegativeReals) # Tons of corn purchased
model.Ws = Var(domain=NonNegativeReals) # Tons of wheat sold
model.Cs = Var(domain=NonNegativeReals) # Tons of corn sold
model.Sq = Var(domain=NonNegativeReals) # Tons of sugar beets sold at quota price
model.Se = Var(domain=NonNegativeReals) # Tons of sugar beets sold in excess of quota

Define objective function
model.obj = Objective(expr=150*model.W + 230*model.C + 260*model.S + 238*model.Wp + 210*model.Cp - 170*model.Ws - 150*

Define constraints
model.land_constraint = Constraint(expr=model.W + model.C + model.S <= 500)
model.wheat_constraint = Constraint(expr=2.5*model.W + model.Wp >= 200)
model.corn_constraint = Constraint(expr=3*model.C + model.Cp >= 240)
model.wheat_balance = Constraint(expr=2.5*model.W == 200 + model.Ws)
model.corn_balance = Constraint(expr=3*model.C == 240 + model.Cs)
model.sugar_beet_quota = Constraint(expr=model.Sq <= 6000)
model.sugar_beet_balance = Constraint(expr=20*model.S == model.Sq + model.Se)

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

Print the results
print("Objective Function Value:", model.obj())
print("Acres of Wheat (W):", model.W())
print("Acres of Corn (C):", model.C())
print("Acres of Sugar Beets (S):", model.S())
print("Tons of Wheat Purchased (Wp):", model.Wp())
print("Tons of Corn Purchased (Cp):", model.Cp())
print("Tons of Wheat Sold (Ws):", model.Ws())
print("Tons of Corn Sold (Cs):", model.Cs())
print("Tons of Sugar Beets Sold at Quota Price (Sq):", model.Sq())
print("Tons of Sugar Beets Sold in Excess of Quota (Se):", model.Se())
```

```

In this code:

1. We create a concrete model using Pyomo.
2. We define the decision variables (W, C, S, Wp, Cp, Ws, Cs, Sq, Se) as non-negative real numbers.
3. We define the objective function, which is to minimize the total costs based on the given cost coefficients and decision variables.
4. We define the constraints using the expressions provided in the previous response.
5. We solve the model using the 'glpk' solver (you can change it to any other solver available in your environment).
6. Finally, we print the results, including the objective function value and the values of the decision variables.

Note: This code assumes that you have Pyomo and a suitable solver (e.g., GLPK) installed in your Python environment. Y

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

```

from pyomo.environ import *

```

```

# Create a Concrete Model
model = ConcreteModel()

# Define decision variables
model.W = Var(domain=NonNegativeReals) # Acres of land for wheat
model.C = Var(domain=NonNegativeReals) # Acres of land for corn
model.S = Var(domain=NonNegativeReals) # Acres of land for sugar beets
model.Wp = Var(domain=NonNegativeReals) # Tons of wheat purchased
model.Cp = Var(domain=NonNegativeReals) # Tons of corn purchased
model.Ws = Var(domain=NonNegativeReals) # Tons of wheat sold
model.Cs = Var(domain=NonNegativeReals) # Tons of corn sold
model.Sq = Var(domain=NonNegativeReals) # Tons of sugar beets sold at quota price
model.Se = Var(domain=NonNegativeReals) # Tons of sugar beets sold in excess of quota


# Define objective function
model.obj = Objective(expr=150*model.W + 230*model.C + 260*model.S + 238*model.Wp + 210*model.Cp - 170*model.Ws - 150*model.Cs)

# Define constraints
model.land_constraint = Constraint(expr=model.W + model.C + model.S <= 500)
model.wheat_constraint = Constraint(expr=2.5*model.W + model.Wp >= 200)
model.corn_constraint = Constraint(expr=3*model.C + model.Cp >= 240)
model.wheat_balance = Constraint(expr=2.5*model.W == 200 + model.Ws)
model.corn_balance = Constraint(expr=3*model.C == 240 + model.Cs)
model.sugar_beet_quota = Constraint(expr=model.Sq <= 6000)
model.sugar_beet_balance = Constraint(expr=20*model.S == model.Sq + model.Se)

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

# Print the results
print("Objective Function Value:", model.obj())
print("Acres of Wheat (W):", model.W())
print("Acres of Corn (C):", model.C())
print("Acres of Sugar Beets (S):", model.S())
print("Tons of Wheat Purchased (Wp):", model.Wp())
print("Tons of Corn Purchased (Cp):", model.Cp())
print("Tons of Wheat Sold (Ws):", model.Ws())
print("Tons of Corn Sold (Cs):", model.Cs())
print("Tons of Sugar Beets Sold at Quota Price (Sq):", model.Sq())
print("Tons of Sugar Beets Sold in Excess of Quota (Se):", model.Se())

```

 Objective Function Value: -118599.99999999999
 Acres of Wheat (W): 120.0
 Acres of Corn (C): 80.0
 Acres of Sugar Beets (S): 300.0
 Tons of Wheat Purchased (Wp): 0.0
 Tons of Corn Purchased (Cp): 0.0
 Tons of Wheat Sold (Ws): 99.99999999999999
 Tons of Corn Sold (Cs): 0.0
 Tons of Sugar Beets Sold at Quota Price (Sq): 6000.0
 Tons of Sugar Beets Sold in Excess of Quota (Se): 0.0

✓ 5. Correct The Model Code to Test Mathematical Model (if applicable)