

✓ 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/IP/IP4.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:
 Imagine you own a small shop, and you're trying to manage your employees' working hours and sales targets efficiently. You have three kinds of goals overall: sales goals, employee hours goals and overtime goals.

Sales Goals:
 You have a target of selling 5500 units.
 But sometimes, you might sell fewer (negative deviation) or more (positive deviation) than this target.
 On average, your full-time employees will sell 5 records per hour and your part-time employees will sell 2 records per h

Employee Hours:
 You have both full-time and part-time employees.
 Your full-time employees are expected to work a total of 800 hours, while part-time employees are expected to work 320 h
 Similarly, like sales, there could be deviations in these working hours.

Overtime:

Additionally, your employees might end up working overtime.
You have a target of 100 hours of overtime for full-time employees.
You are not concerned about limiting the overtime of part-time employees.
Again, you might deviate from this goal.

Each of these deviations has a different priority for you.

P1 to P4 are priority factors for each deviation.

Your most important goal is to achieve your sales goal.

Second to that is the goal to not exceed 100 overtime hours for full-time employees.

Then, your next priority is to fully utilize your full-time and part time employees.

Your last goal with the lowest priority, you want to keep general overtime hours for both kinds of employees as low as possible.

In addition to the general priorities of your goals, you are twice as concerned about fully utilizing the full-time employees.

Finally, overtime hours of part-time employees have a relative cost of three times that of full-time employees.

Now, the goal here is to minimize the overall impact of these deviations, considering their priorities.

This means finding the best balance between sales targets, employee hours, and overtime, while keeping in mind which deviations are most important.

```
!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)
```

 Here are the variables for the mathematical optimization problem:

Decision Variables:

- F: Number of hours worked by full-time employees
- P: Number of hours worked by part-time employees

Deviation Variables:

- d1+: Positive deviation from sales target (units)
- d1-: Negative deviation from sales target (units)
- d2+: Positive deviation from full-time employee hours target
- d2-: Negative deviation from full-time employee hours target
- d3+: Positive deviation from part-time employee hours target
- d3-: Negative deviation from part-time employee hours target
- d4+: Positive deviation from full-time employee overtime hours target
- d4-: Negative deviation from full-time employee overtime hours target
- d5+: Positive deviation from total overtime hours (for both full-time and part-time employees)

Priority Factors:

- P1: Priority for deviations from sales target
- P2: Priority for deviations from full-time employee overtime hours target
- P3: Priority for deviations from full-time employee hours target
- P4: Priority for deviations from part-time employee hours target
- P5: Priority for deviations from total overtime hours

Constants:

- 5500: Sales target (units)
- 5: Sales per hour for full-time employees (units/hour)
- 2: Sales per hour for part-time employees (units/hour)
- 800: Full-time employee hours target
- 320: Part-time employee hours target
- 100: Full-time employee overtime hours target
- 2: Relative importance of utilizing full-time employee hours compared to part-time employee hours
- 3: Relative cost of part-time employee overtime hours compared to full-time employee overtime hours

✓ 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)
```

🔄 Objective Function:

Minimize:

$$P1 * (d1+ + d1-) + P2 * (d4+ + d4-) + P3 * 2 * (d2+ + d2-) + P4 * (d3+ + d3-) + P5 * (d5+ + 3 * (P - 320))$$

Explanation:

- $P1 * (d1+ + d1-)$: Minimizes the deviations from the sales target, with priority P1.
- $P2 * (d4+ + d4-)$: Minimizes the deviations from the full-time employee overtime hours target, with priority P2.
- $P3 * 2 * (d2+ + d2-)$: Minimizes the deviations from the full-time employee hours target, with priority P3. The factor
- $P4 * (d3+ + d3-)$: Minimizes the deviations from the part-time employee hours target, with priority P4.
- $P5 * (d5+ + 3 * (P - 320))$: Minimizes the total overtime hours for both full-time and part-time employees, with priori

Note: The decision variables F and P, along with the other deviation variables, will be used in the constraints of the o

✓ 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:

Sales Constraint:

$$5 * F + 2 * P + d1- - d1+ = 5500$$

Explanation:

This constraint ensures that the total sales achieved by full-time and part-time employees, plus any deviations, meet th

Full-time Employee Hours Constraint:

$$F + d2- - d2+ = 800$$

Explanation:

This constraint ensures that the total hours worked by full-time employees, plus any deviations, meet the full-time empl

Part-time Employee Hours Constraint:

$$P + d3- - d3+ = 320$$

Explanation:

This constraint ensures that the total hours worked by part-time employees, plus any deviations, meet the part-time empl

Full-time Employee Overtime Hours Constraint:

$$F - 800 + d4- - d4+ = 100$$

Explanation:

This constraint ensures that the overtime hours worked by full-time employees, plus any deviations, meet the full-time e

Total Overtime Hours Constraint:

$$(F - 800) + (P - 320) + d5- - d5+ = 0$$

Explanation:

This constraint ensures that the total overtime hours worked by both full-time and part-time employees, plus any deviati

Non-negativity Constraints:

$F, P, d1+, d1-, d2+, d2-, d3+, d3-, d4+, d4-, d5+ \geq 0$

Explanation:

These constraints ensure that all decision variables and deviation variables are non-negative.

By combining the objective function and these constraints, you have a complete mathematical optimization problem that ai

✓ 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=4096,
    messages=[
        {"role": "user", "content": (response_p11 + response_p12 + response_p13)}
    ]
)

response_p2 = message2.content[0].text

# Print response
print(response_p2)
```

```

python
from pyomo.environ import *

# Create the model
model = ConcreteModel()

# Define the decision variables
model.F = Var(within=NonNegativeReals) # Number of hours worked by full-time employees
model.P = Var(within=NonNegativeReals) # Number of hours worked by part-time employees

# Define the deviation variables
model.d1_plus = Var(within=NonNegativeReals) # Positive deviation from sales target
model.d1_minus = Var(within=NonNegativeReals) # Negative deviation from sales target
model.d2_plus = Var(within=NonNegativeReals) # Positive deviation from full-time employee hours target
model.d2_minus = Var(within=NonNegativeReals) # Negative deviation from full-time employee hours target
model.d3_plus = Var(within=NonNegativeReals) # Positive deviation from part-time employee hours target
model.d3_minus = Var(within=NonNegativeReals) # Negative deviation from part-time employee hours target
model.d4_plus = Var(within=NonNegativeReals) # Positive deviation from full-time employee overtime hours target
model.d4_minus = Var(within=NonNegativeReals) # Negative deviation from full-time employee overtime hours target
model.d5_plus = Var(within=NonNegativeReals) # Positive deviation from total overtime hours

# Define the priority factors (sample data)
P1 = 10 # Priority for deviations from sales target
P2 = 8 # Priority for deviations from full-time employee overtime hours target
P3 = 6 # Priority for deviations from full-time employee hours target
P4 = 4 # Priority for deviations from part-time employee hours target
P5 = 2 # Priority for deviations from total overtime hours

# Define the objective function
model.objective = Objective(expr=P1 * (model.d1_plus + model.d1_minus) +
                             P2 * (model.d4_plus + model.d4_minus) +
                             P3 * 2 * (model.d2_plus + model.d2_minus) +
                             P4 * (model.d3_plus + model.d3_minus) +
                             P5 * (model.d5_plus + 3 * (model.P - 320)),
                             sense=minimize)

# Define the constraints
model.sales_constraint = Constraint(expr=5 * model.F + 2 * model.P + model.d1_minus - model.d1_plus == 5500)
model.fulltime_hours_constraint = Constraint(expr=model.F + model.d2_minus - model.d2_plus == 800)
model.parttime_hours_constraint = Constraint(expr=model.P + model.d3_minus - model.d3_plus == 320)
model.fulltime_overtime_constraint = Constraint(expr=model.F - 800 + model.d4_minus - model.d4_plus == 100)
model.total_overtime_constraint = Constraint(expr=(model.F - 800) + (model.P - 320) + model.d5_minus - model.d5_plus == 0)

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

```
print("Objective value:", model.objective())
print("Full-time employee hours (F):", model.F())
print("Part-time employee hours (P):", model.P())
`
```

Note: The priority factors (P1, P2, P3, P4, P5) are sample data and should be adjusted based on the actual problem req

This code creates a Pyomo model with the decision variables, deviation variables, objective function, and constraints

Please make sure you have Pyomo and the `glpk` solver installed before running this 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

```
from pyomo.environ import *

# Create the model
model = ConcreteModel()

# Define the decision variables
model.F = Var(within=NonNegativeReals) # Number of hours worked by full-time employees
model.P = Var(within=NonNegativeReals) # Number of hours worked by part-time employees

# Define the deviation variables
model.d1_plus = Var(within=NonNegativeReals) # Positive deviation from sales target
model.d1_minus = Var(within=NonNegativeReals) # Negative deviation from sales target
model.d2_plus = Var(within=NonNegativeReals) # Positive deviation from full-time employee hours target
model.d2_minus = Var(within=NonNegativeReals) # Negative deviation from full-time employee hours target
model.d3_plus = Var(within=NonNegativeReals) # Positive deviation from part-time employee hours target
model.d3_minus = Var(within=NonNegativeReals) # Negative deviation from part-time employee hours target
model.d4_plus = Var(within=NonNegativeReals) # Positive deviation from full-time employee overtime hours target
model.d4_minus = Var(within=NonNegativeReals) # Negative deviation from full-time employee overtime hours target
model.d5_plus = Var(within=NonNegativeReals) # Positive deviation from total overtime hours

# Define the priority factors (sample data)
P1 = 5 # Priority for deviations from sales target
P2 = 4 # Priority for deviations from full-time employee overtime hours target
P3 = 3 # Priority for deviations from full-time employee hours target
P4 = 2 # Priority for deviations from part-time employee hours target
P5 = 1 # Priority for deviations from total overtime hours

# Define the objective function
model.objective = Objective(expr=P1 * (model.d1_plus + model.d1_minus) +
                             P2 * (model.d4_plus + model.d4_minus) +
                             P3 * 2 * (model.d2_plus + model.d2_minus) +
                             P4 * (model.d3_plus + model.d3_minus) +
                             P5 * (model.d5_plus + 3 * (model.P - 320)),
                             sense=minimize)

# Define the constraints
model.sales_constraint = Constraint(expr=5 * model.F + 2 * model.P + model.d1_minus - model.d1_plus == 5500)
model.fulltime_hours_constraint = Constraint(expr=model.F + model.d2_minus - model.d2_plus == 800)
model.parttime_hours_constraint = Constraint(expr=model.P + model.d3_minus - model.d3_plus == 320)
model.fulltime_overtime_constraint = Constraint(expr=model.F - 800 + model.d4_minus - model.d4_plus == 100)
model.total_overtime_constraint = Constraint(expr=(model.F - 800) + (model.P - 320) + model.d5_minus - model.d5_plus == 0)

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

# Print the results
print("Objective value:", model.objective())
print("Full-time employee hours (F):", model.F())
print("Part-time employee hours (P):", model.P())
```

```

-----
AttributeError                                Traceback (most recent call last)
<ipython-input-16-70d48aa4d620> in <cell line: 41>()
    39 model.parttime_hours_constraint = Constraint(expr=model.P +
model.d3_minus - model.d3_plus == 320)
    40 model.fulltime_overtime_constraint = Constraint(expr=model.F - 800 +
model.d4_minus - model.d4_plus == 100)
--> 41 model.total_overtime_constraint = Constraint(expr=(model.F - 800) +
(model.P - 320) + model.d5_minus - model.d5_plus == 0)
    42
    43 # Solve the model

/usr/local/lib/python3.10/dist-packages/pyomo/core/base/block.py in
_getattr__(self, val)
    548     # Since the base classes don't support getattr, we can just
    549     # throw the "normal" AttributeError
--> 550     raise AttributeError(
    551         "'%s' object has no attribute '%s'" %
(self, class . name . val)

print("d1^-:", model.d1_minus())
print("d1^+:", model.d1_plus())
print("d2^-:", model.d2_minus())
print("d2^+:", model.d2_plus())
print("d3^-:", model.d3_minus())
print("d3^+:", model.d3_plus())
print("d4^-:", model.d4_minus())
print("d4^+:", model.d4_plus())
print("d5^+:", model.d5_plus())

```

```

d1^-: 0.0
d1^+: 0.0
d2^-: 0.0
d2^+: 172.0
d3^-: 0.0
d3^+: 0.0
d4^-: 0.0
d4^+: 72.0
d5^+: 172.0

```

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

```

from pyomo.environ import *

# Create the model
model = ConcreteModel()

# Define the decision variables
model.F = Var(within=NonNegativeReals) # Number of hours worked by full-time employees
model.P = Var(within=NonNegativeReals) # Number of hours worked by part-time employees

# Define the deviation variables
model.d1_plus = Var(within=NonNegativeReals) # Positive deviation from sales target
model.d1_minus = Var(within=NonNegativeReals) # Negative deviation from sales target
model.d2_plus = Var(within=NonNegativeReals) # Positive deviation from full-time employee hours target
model.d2_minus = Var(within=NonNegativeReals) # Negative deviation from full-time employee hours target
model.d3_plus = Var(within=NonNegativeReals) # Positive deviation from part-time employee hours target
model.d3_minus = Var(within=NonNegativeReals) # Negative deviation from part-time employee hours target
model.d4_plus = Var(within=NonNegativeReals) # Positive deviation from full-time employee overtime hours target
model.d4_minus = Var(within=NonNegativeReals) # Negative deviation from full-time employee overtime hours target
model.d5_plus = Var(within=NonNegativeReals) # Positive deviation from total overtime hours
model.d5_minus = Var(within=NonNegativeReals) # Positive deviation from total overtime hours

# Define the priority factors (sample data)
P1 = 5 # Priority for deviations from sales target
P2 = 4 # Priority for deviations from full-time employee overtime hours target
P3 = 3 # Priority for deviations from full-time employee hours target
P4 = 2 # Priority for deviations from part-time employee hours target
P5 = 1 # Priority for deviations from total overtime hours

# Define the objective function
model.objective = Objective(expr=P1 * (model.d1_plus + model.d1_minus) +
                             P2 * (model.d4_plus + model.d4_minus) +
                             P3 * 2 * (model.d2_plus + model.d2_minus) +
                             P4 * (model.d3_plus + model.d3_minus) +
                             P5 * (model.d5_plus + 3 * (model.P - 320)),
                             sense=minimize)

# Define the constraints
model.sales_constraint = Constraint(expr=5 * model.F + 2 * model.P + model.d1_minus - model.d1_plus == 5500)
model.fulltime_hours_constraint = Constraint(expr=model.F + model.d2_minus - model.d2_plus == 800)
model.parttime_hours_constraint = Constraint(expr=model.P + model.d3_minus - model.d3_plus == 320)

```

```
model.fulltime_overtime_constraint = Constraint(expr=model.F - 800 + model.d4_minus - model.d4_plus == 100)
model.total_overtime_constraint = Constraint(expr=(model.F - 800) + (model.P - 320) + model.d5_minus - model.d5_plus == 0)

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

# Print the results
print("Objective value:", model.objective())
print("Full-time employee hours (F):", model.F())
print("Part-time employee hours (P):", model.P())
print("d1^-:", model.d1_minus())
print("d1^+:", model.d1_plus())
print("d2^-:", model.d2_minus())
print("d2^+:", model.d2_plus())
print("d3^-:", model.d3_minus())
print("d3^+:", model.d3_plus())
print("d4^-:", model.d4_minus())
print("d4^+:", model.d4_plus())
print("d5^-:", model.d5_minus())
print("d5^+:", model.d5_plus())
```

```
➦ Objective value: 1492.0
Full-time employee hours (F): 972.0
Part-time employee hours (P): 320.0
d1^-: 0.0
d1^+: 0.0
d2^-: 0.0
d2^+: 172.0
d3^-: 0.0
d3^+: 0.0
d4^-: 0.0
d4^+: 72.0
d5^-: 172.0
```