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 (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:
 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, you 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 p
In addition to the general priorities of you goals, you are twice as concerned about fully utilizing the full-time emplo
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 dev
```

```
!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)
```

```
Okay, let's formulate the variables for this goal programming problem.

Decision Variables:
- Let FT represent the number of hours worked by full-time employees
- Let PT represent the number of hours worked by part-time employees

Deviation Variables:
- Let d1- represent the negative deviation from the sales target (underachievement)
- Let d1+ represent the positive deviation from the sales target (overachievement)
- Let d2- represent the negative deviation from the full-time employee hours target (underutilization)
- Let d2+ represent the positive deviation from the full-time employee hours target (overutilization)
- Let d3- represent the negative deviation from the part-time employee hours target (underutilization)
- Let d3+ represent the positive deviation from the part-time employee hours target (overutilization)
- Let d4- represent the negative deviation from the overtime target for full-time employees (underutilization)
- Let d4+ represent the positive deviation from the overtime target for full-time employees (overutilization)
- Let d5- represent the total overtime hours worked by full-time employees
- Let d6- represent the total overtime hours worked by part-time employees

Priority Factors:
- Let P1 represent the priority for the sales goal
- Let P2 represent the priority for the full-time employee overtime goal
- Let P3 represent the priority for the full-time employee hours goal
- Let P4 represent the priority for the part-time employee hours goal and the general overtime goal

Note: The priority factors are ordered from highest to lowest, i.e., P1 > P2 > P3 > P4.

These variables will be used to formulate the constraints and the objective function of the goal programming problem, wh
```

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 given information, the objective function for this goal programming problem can be formulated as:

Minimize:
P1(d1- + d1+) + P2(d4+) + P3(2d2- + 2d2+ + d3- + d3+) + P4(d5- + 3d6-)

Explanation:
- P1(d1- + d1+): This term represents the priority of achieving the sales target. Both underachievement (d1-) and overac
- P2(d4+): This term represents the priority of not exceeding the overtime target for full-time employees. Only the posi
- P3(2d2- + 2d2+ + d3- + d3+): This term represents the priority of fully utilizing both full-time and part-time employe
- P4(d5- + 3d6-): This term represents the priority of minimizing the overall overtime hours for both full-time (d5-) an

The objective function aims to minimize the weighted sum of the deviations based on their priorities, considering the sp
```

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)
```

```
Great! Let's formulate the constraints for this goal programming problem based on the given information.

Constraints:

1. Sales Target:
   5FT + 2PT + d1- - d1+ = 5500

   Explanation: This constraint ensures that the total sales achieved by full-time (5 units per hour) and part-time (2 u

2. Full-time Employee Hours:
   FT + d2- - d2+ = 800

   Explanation: This constraint ensures that the total hours worked by full-time employees, along with any deviations, m

3. Part-time Employee Hours:
   PT + d3- - d3+ = 320

   Explanation: This constraint ensures that the total hours worked by part-time employees, along with any deviations, m

4. Overtime for Full-time Employees:
   FT - 800 + d4- - d4+ = 100

   Explanation: This constraint ensures that the overtime hours for full-time employees, calculated as the difference be

5. Total Overtime for Full-time Employees:
   FT - 800 - d5- = 0

   Explanation: This constraint captures the total overtime hours worked by full-time employees, calculated as the diffe

6. Total Overtime for Part-time Employees:
   PT - 320 - d6- = 0

   Explanation: This constraint captures the total overtime hours worked by part-time employees, calculated as the diffe

7. Non-negativity Constraints:
   FT, PT, d1-, d1+, d2-, d2+, d3-, d3+, d4-, d4+, d5-, d6- ≥ 0

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

```
These constraints, along with the objective function, complete the formulation of the goal programming problem. The obje
```

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)
```

```
# Define decision variables
model.FT = Var(domain=NonNegativeReals)
model.PT = Var(domain=NonNegativeReals)

# Define deviation variables
model.d1_minus = Var(domain=NonNegativeReals)
model.d1_plus = Var(domain=NonNegativeReals)
model.d2_minus = Var(domain=NonNegativeReals)
model.d2_plus = Var(domain=NonNegativeReals)
model.d3_minus = Var(domain=NonNegativeReals)
model.d3_plus = Var(domain=NonNegativeReals)
model.d4_minus = Var(domain=NonNegativeReals)
model.d4_plus = Var(domain=NonNegativeReals)
model.d5_minus = Var(domain=NonNegativeReals)
model.d6_minus = Var(domain=NonNegativeReals)

# Define priority factors (sample data)
P1 = 1000
P2 = 100
P3 = 10
P4 = 1

# Define the objective function
model.objective = Objective(expr=P1 * (model.d1_minus + model.d1_plus) +
                                 P2 * (model.d4_plus) +
                                 P3 * (2 * model.d2_minus + 2 * model.d2_plus + model.d3_minus + model.d3_plus) +
                                 P4 * (model.d5_minus + 3 * model.d6_minus),
                            sense=minimize)

# Define constraints
model.sales_constraint = Constraint(expr=5 * model.FT + 2 * model.PT + model.d1_minus - model.d1_plus == 5500)
model.full_time_hours_constraint = Constraint(expr=model.FT + model.d2_minus - model.d2_plus == 800)
model.part_time_hours_constraint = Constraint(expr=model.PT + model.d3_minus - model.d3_plus == 320)
model.full_time_overtime_constraint = Constraint(expr=model.FT - 800 + model.d4_minus - model.d4_plus == 100)
model.full_time_total_overtime_constraint = Constraint(expr=model.FT - 800 - model.d5_minus == 0)
model.part_time_total_overtime_constraint = Constraint(expr=model.PT - 320 - model.d6_minus == 0)

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

# Print the results
print("Objective Function Value:", model.objective())
print("Full-time Employee Hours (FT):", model.FT())
print("Part-time Employee Hours (PT):", model.PT())
```

```
- Finally, we print the objective function value and the optimal values of the decision variables (`FT` and `PT`).

Note: Make sure you have the Pyomo library installed (`pip install pyomo`) before running this code.
```

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

```
from pyomo.environ import *

# Create a Pyomo model
model = ConcreteModel()

# Define decision variables
model.FT = Var(domain=NonNegativeReals)
model.PT = Var(domain=NonNegativeReals)

# Define deviation variables
model.d1_minus = Var(domain=NonNegativeReals)
model.d1_plus = Var(domain=NonNegativeReals)
model.d2_minus = Var(domain=NonNegativeReals)
model.d2_plus = Var(domain=NonNegativeReals)
model.d3_minus = Var(domain=NonNegativeReals)
model.d3_plus = Var(domain=NonNegativeReals)
model.d4_minus = Var(domain=NonNegativeReals)
model.d4_plus = Var(domain=NonNegativeReals)
model.d5_minus = Var(domain=NonNegativeReals)
model.d6_minus = Var(domain=NonNegativeReals)

# Define priority factors (sample data)
P1 = 4
P2 = 3
P3 = 2
P4 = 1

# Define the objective function
model.objective = Objective(expr=P1 * (model.d1_minus + model.d1_plus) +
                                 P2 * (model.d4_plus) +
                                 P3 * (2 * model.d2_minus + 2 * model.d2_plus + model.d3_minus + model.d3_plus) +
                                 P4 * (model.d5_minus + 3 * model.d6_minus),
                            sense=minimize)

# Define constraints
model.sales_constraint = Constraint(expr=5 * model.FT + 2 * model.PT + model.d1_minus - model.d1_plus == 5500)
model.full_time_hours_constraint = Constraint(expr=model.FT + model.d2_minus - model.d2_plus == 800)
model.part_time_hours_constraint = Constraint(expr=model.PT + model.d3_minus - model.d3_plus == 320)
model.full_time_overtime_constraint = Constraint(expr=model.FT - 800 + model.d4_minus - model.d4_plus == 100)
model.full_time_total_overtime_constraint = Constraint(expr=model.FT - 800 - model.d5_minus == 0)
model.part_time_total_overtime_constraint = Constraint(expr=model.PT - 320 - model.d6_minus == 0)

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

# Print the results
print("Objective Function Value:", model.objective())
print("Full-time Employee Hours (FT):", model.FT())
print("Part-time Employee Hours (PT):", model.PT())
```

```
Objective Function Value: 1076.0
Full-time Employee Hours (FT): 972.0
Part-time Employee Hours (PT): 320.0
```

```
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("d6^-:", model.d6_minus())
```

```
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
d6^-: -0.0
```

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