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/IP/IP4.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:
 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: 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)
```

```
Great! Let's formulate this problem as a mathematical optimization model. We will use the goal programming approach to h

Parameters:
- ST: Sales Target (5500 units)
- FTH: Full-Time employee Hours goal (800 hours)
- PTH: Part-Time employee Hours goal (320 hours)
- FTOT: Full-Time employee Overtime Target (100 hours)
- FTS: Full-Time employee Sales per hour (5 units/hour)
- PTS: Part-Time employee Sales per hour (2 units/hour)
- P1, P2, P3, P4: Priority factors for deviations

Decision Variables:
- FT: Full-Time employee hours
- PT: Part-Time employee hours
- FTOV: Full-Time employee Overtime hours
- PTOV: Part-Time employee Overtime hours
- S⁻, S⁺: Negative and positive deviations from the Sales Target
- FTH⁻, FTH⁺: Negative and positive deviations from the Full-Time employee Hours goal
- PTH⁻, PTH⁺: Negative and positive deviations from the Part-Time employee Hours goal
- FTOT⁻, FTOT⁺: Negative and positive deviations from the Full-Time employee Overtime Target

Objective Function:
Minimize Z = P1 * (S⁻ + S⁺) + P2 * (FTOT⁻ + FTOT⁺) + P3 * (2 * FTH⁻ + 2 * FTH⁺ + PTH⁻ + PTH⁺) + P4 * (FTOV + 3 * PTOV)

Constraints:
1. Sales Constraint:
   FTS * FT + PTS * PT + S⁻ - S⁺ = ST

2. Full-Time Hours Constraint:
   FT + FTH⁻ - FTH⁺ = FTH

3. Part-Time Hours Constraint:
   PT + PTH⁻ - PTH⁺ = PTH

4. Full-Time Overtime Constraint:
   FTOV + FTOT⁻ - FTOT⁺ = FTOT

5. Non-negativity Constraints:
   FT, PT, FTOV, PTOV, S⁻, S⁺, FTH⁻, FTH⁺, PTH⁻, PTH⁺, FTOT⁻, FTOT⁺ ≥ 0

The objective function minimizes the weighted sum of deviations based on their priorities. The constraints ensure that t

By solving this goal programming model, you can find the optimal balance between sales targets, employee hours, and over
```

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.FT = Var(domain=NonNegativeReals)
model.PT = Var(domain=NonNegativeReals)
model.FTOV = Var(domain=NonNegativeReals)
model.PTOV = Var(domain=NonNegativeReals)
model.S_neg = Var(domain=NonNegativeReals)
model.S_pos = Var(domain=NonNegativeReals)
model.FTH_neg = Var(domain=NonNegativeReals)
model.FTH_pos = Var(domain=NonNegativeReals)
model.PTH_neg = Var(domain=NonNegativeReals)
model.PTH_pos = Var(domain=NonNegativeReals)
model.FTOT_neg = Var(domain=NonNegativeReals)
model.FTOT_pos = Var(domain=NonNegativeReals)

# Define the objective function
def objective_rule(model):
    return (
        model.P1 * (model.S_neg + model.S_pos) +
        model.P2 * (model.FTOT_neg + model.FTOT_pos) +
        model.P3 * (2 * model.FTH_neg + 2 * model.FTH_pos + model.PTH_neg + model.PTH_pos) +
        model.P4 * (model.FTOV + 3 * model.PTOV)
    )
model.objective = Objective(rule=objective_rule, sense=minimize)

# Define the constraints
def sales_constraint(model):
    return model.FTS * model.FT + model.PTS * model.PT + model.S_neg - model.S_pos == model.ST
model.sales_constraint = Constraint(rule=sales_constraint)

def fulltime_hours_constraint(model):
    return model.FT + model.FTH_neg - model.FTH_pos == model.FTH
model.fulltime_hours_constraint = Constraint(rule=fulltime_hours_constraint)

def parttime_hours_constraint(model):
    return model.PT + model.PTH_neg - model.PTH_pos == model.PTH
model.parttime_hours_constraint = Constraint(rule=parttime_hours_constraint)

def fulltime_overtime_constraint(model):
    return model.FTOV + model.FTOT_neg - model.FTOT_pos == model.FTOT
model.fulltime_overtime_constraint = Constraint(rule=fulltime_overtime_constraint)

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

# Print the results
print("Objective Function Value:", model.objective())
print("Full-Time Hours:", model.FT())
print("Part-Time Hours:", model.PT())
print("Full-Time Overtime Hours:", model.FTOV())
print("Part-Time Overtime Hours:", model.PTOV())
```

In this code, we define the parameters using sample data. The priority factors (P1, P2, P3, P4) are also assigned samp

The decision variables are defined as non-negative real variables using `Var(domain=NonNegativeReals)`.

The objective function and constraints are defined as rules using Python functions. The objective function minimizes t

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 the model
model = ConcreteModel()

# Define the parameters (sample data)
model.ST = Param(initialize=5500)
model.FTH = Param(initialize=800)
model.PTH = Param(initialize=320)
model.FTOT = Param(initialize=100)
model.FTS = Param(initialize=5)
model.PTS = Param(initialize=2)
model.P1 = Param(initialize=4)  # Sample priority factor
model.P2 = Param(initialize=3)   # Sample priority factor
model.P3 = Param(initialize=2)   # Sample priority factor
model.P4 = Param(initialize=1)   # Sample priority factor

# Define the decision variables
model.FT = Var(domain=NonNegativeReals)
model.PT = Var(domain=NonNegativeReals)
model.FTOV = Var(domain=NonNegativeReals)
model.PTOV = Var(domain=NonNegativeReals)
model.S_neg = Var(domain=NonNegativeReals)
model.S_pos = Var(domain=NonNegativeReals)
model.FTH_neg = Var(domain=NonNegativeReals)
model.FTH_pos = Var(domain=NonNegativeReals)
model.PTH_neg = Var(domain=NonNegativeReals)
model.PTH_pos = Var(domain=NonNegativeReals)
model.FTOT_neg = Var(domain=NonNegativeReals)
model.FTOT_pos = Var(domain=NonNegativeReals)

# Define the objective function
def objective_rule(model):
    return (
        model.P1 * (model.S_neg + model.S_pos) +
        model.P2 * (model.FTOT_neg + model.FTOT_pos) +
        model.P3 * (2 * model.FTH_neg + 2 * model.FTH_pos + model.PTH_neg + model.PTH_pos) +
        model.P4 * (model.FTOV + 3 * model.PTOV)
    )
model.objective = Objective(rule=objective_rule, sense=minimize)

# Define the constraints
def sales_constraint(model):
    return model.FTS * model.FT + model.PTS * model.PT + model.S_neg - model.S_pos == model.ST
model.sales_constraint = Constraint(rule=sales_constraint)

def fulltime_hours_constraint(model):
    return model.FT + model.FTH_neg - model.FTH_pos == model.FTH
model.fulltime_hours_constraint = Constraint(rule=fulltime_hours_constraint)

def parttime_hours_constraint(model):
    return model.PT + model.PTH_neg - model.PTH_pos == model.PTH
model.parttime_hours_constraint = Constraint(rule=parttime_hours_constraint)

def fulltime_overtime_constraint(model):
    return model.FTOV + model.FTOT_neg - model.FTOT_pos == model.FTOT
model.fulltime_overtime_constraint = Constraint(rule=fulltime_overtime_constraint)

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

# Print the results
print("Objective Function Value:", model.objective())
print("Full-Time Hours:", model.FT())
print("Part-Time Hours:", model.PT())
print("Full-Time Overtime Hours:", model.FTOV())
print("Part-Time Overtime Hours:", model.PTOV())
```

```
Objective Function Value: 788.0
Full-Time Hours: 972.0
Part-Time Hours: 320.0
Full-Time Overtime Hours: 100.0
Part-Time Overtime Hours: 0.0
```

```
print(model.FT())
print(model.PT())
print(model.FTOV())
print(model.PTOV())
print(model.S_neg())
print(model.S_pos())
print(model.FTH_neg())
print(model.FTH_pos())
print(model.PTH_neg())
print(model.PTH_pos())
print(model.FTOT_neg())
print(model.FTOT_pos())
```

```
972.0
320.0
100.0
0.0
0.0
0.0
0.0
172.0
0.0
0.0
0.0
0.0
```

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