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/IP3.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 person in charge of packing in a large company. Your job is to skillfully pack items of various weights in
```

```
!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 formulate a mathematical optimization model for the packing problem, we will define the parameters, decision variable

Parameters:
- N: The total number of items to be packed
- M: The maximum number of boxes available
- w_i: The weight of item i, where i = 1, 2, ..., N
- C: The capacity of each box (assuming all boxes have the same capacity)

Decision Variables:
- x_ij: Binary variable; 1 if item i is packed into box j, 0 otherwise, where i = 1, 2, ..., N and j = 1, 2, ..., M
- y_j: Binary variable; 1 if box j is used, 0 otherwise, where j = 1, 2, ..., M

Objective Function:
- Minimize: ∑(j=1 to M) y_j
  - The objective is to minimize the total number of boxes used.

Constraints:
1. Each item must be packed into exactly one box:
   - ∑(j=1 to M) x_ij = 1, for all i = 1, 2, ..., N

2. The total weight of items packed into each box must not exceed the box capacity:
   - ∑(i=1 to N) w_i * x_ij ≤ C * y_j, for all j = 1, 2, ..., M
   - This constraint ensures that the sum of weights of items packed into box j does not exceed the capacity C if box j

3. Ensure that x_ij can only be 1 if box j is used:
   - x_ij ≤ y_j, for all i = 1, 2, ..., N and j = 1, 2, ..., M
   - This constraint ensures that an item can only be packed into a box if that box is being used.

4. Binary constraints:
   - x_ij ∈ {0, 1}, for all i = 1, 2, ..., N and j = 1, 2, ..., M
   - y_j ∈ {0, 1}, for all j = 1, 2, ..., M

The objective function minimizes the total number of boxes used. Constraint 1 ensures that each item is packed into exac

This mathematical optimization model represents the packing problem and can be solved using integer programming techniqu
```

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)
```

```
Here's the Python Pyomo code for the packing problem optimization model:

```python
from pyomo.environ import *

# Define the model
model = ConcreteModel()

# Define the parameters
model.N = Param(initialize=10)  # Total number of items (sample data)
model.M = Param(initialize=5)   # Maximum number of boxes (sample data)
model.w = Param(RangeSet(1, model.N), initialize={1: 2, 2: 3, 3: 4, 4: 1, 5: 5, 6: 2, 7: 3, 8: 4, 9: 1, 10: 2})  # Wei
model.C = Param(initialize=10)  # Capacity of each box (sample data)

# Define the decision variables
model.x = Var(RangeSet(1, model.N), RangeSet(1, model.M), domain=Binary)
model.y = Var(RangeSet(1, model.M), domain=Binary)

# Define the objective function
model.obj = Objective(expr=sum(model.y[j] for j in range(1, model.M+1)), sense=minimize)

# Define the constraints
def item_packing_rule(model, i):
    return sum(model.x[i, j] for j in range(1, model.M+1)) == 1
model.item_packing = Constraint(RangeSet(1, model.N), rule=item_packing_rule)

def capacity_rule(model, j):
```
```

```
        return sum(model.w[i] * model.x[i, j] for i in range(1, model.N+1)) <= model.C * model.y[j]
    model.capacity = Constraint(RangeSet(1, model.M), rule=capacity_rule)

    def box_usage_rule(model, i, j):
        return model.x[i, j] <= model.y[j]
    model.box_usage = Constraint(RangeSet(1, model.N), RangeSet(1, model.M), rule=box_usage_rule)

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

    # Print the results
    print("Optimization Results:")
    print("Number of boxes used:", value(model.obj))
    print("Packing arrangement:")
    for i in range(1, model.N+1):
        for j in range(1, model.M+1):
            if value(model.x[i, j]) == 1:
                print(f"Item {i} is packed into box {j}")
    ```

    Explanation:
    - We define the model using `ConcreteModel()` from Pyomo.
    - The parameters `N`, `M`, `w`, and `C` are defined using `Param()`. Here, we use sample data:
      - `N = 10` represents the total number of items.
      - `M = 5` represents the maximum number of boxes.
      - `w` is a dictionary that represents the weight of each item.
      - `C = 10` represents the capacity of each box.
    - The decision variables `x` and `y` are defined using `Var()` with binary domains.
    - The objective function is defined using `Objective()`, which minimizes the sum of `y[j]` (number of boxes used)
```

## 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 *

# Define the model
model = ConcreteModel()

# Define the parameters
weights = [2, 2, 2, 2, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6, 7, 7, 8, 8]
model.N = Param(initialize=24)  # Total number of items (sample data)
model.M = Param(initialize=13)   # Maximum number of boxes (sample data)
model.w = Param(RangeSet(1, model.N), initialize={i + 1: weights[i] for i in range(24)})  # Weight of each item (sample data)
model.C = Param(initialize=9)  # Capacity of each box (sample data)

# Define the decision variables
model.x = Var(RangeSet(1, model.N), RangeSet(1, model.M), domain=Binary)
model.y = Var(RangeSet(1, model.M), domain=Binary)

# Define the objective function
model.obj = Objective(expr=sum(model.y[j] for j in range(1, model.M+1)), sense=minimize)

# Define the constraints
def item_packing_rule(model, i):
    return sum(model.x[i, j] for j in range(1, model.M+1)) == 1
model.item_packing = Constraint(RangeSet(1, model.N), rule=item_packing_rule)

def capacity_rule(model, j):
    return sum(model.w[i] * model.x[i, j] for i in range(1, model.N+1)) <= model.C * model.y[j]
model.capacity = Constraint(RangeSet(1, model.M), rule=capacity_rule)

def box_usage_rule(model, i, j):
    return model.x[i, j] <= model.y[j]
model.box_usage = Constraint(RangeSet(1, model.N), RangeSet(1, model.M), rule=box_usage_rule)

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

# Print the results
print("Optimization Results:")
print("Number of boxes used:", value(model.obj))
print("Packing arrangement:")
for i in range(1, model.N+1):
```
```

```
    for j in range(1, model.M+1):
        if value(model.x[i, j]) == 1:
            print(f"Item {i} is packed into box {j}")
```

```
Optimization Results:
Number of boxes used: 13.0
Packing arrangement:
Item 1 is packed into box 11
Item 2 is packed into box 3
Item 3 is packed into box 13
Item 4 is packed into box 13
Item 5 is packed into box 4
Item 6 is packed into box 2
Item 7 is packed into box 8
Item 8 is packed into box 7
Item 9 is packed into box 10
Item 10 is packed into box 5
Item 11 is packed into box 12
Item 12 is packed into box 12
Item 13 is packed into box 13
Item 14 is packed into box 1
Item 15 is packed into box 7
Item 16 is packed into box 10
Item 17 is packed into box 5
Item 18 is packed into box 8
Item 19 is packed into box 2
Item 20 is packed into box 4
Item 21 is packed into box 3
Item 22 is packed into box 11
Item 23 is packed into box 6
Item 24 is packed into box 9
```

```
bins = {1: [], 2: [], 3: [], 4: [], 5: [], 6: [], 7: [], 8: [], 9: [], 10: [], 11: [], 12: [], 13: []}
for (i,j) in model.x:
  if value(model.x[i,j])> .5:
    bins[j].append(model.w[i])

print("Bin Division:", bins)
```

```
Bin Division: {1: [5], 2: [3, 6], 3: [2, 7], 4: [3, 6], 5: [4, 5], 6: [8], 7: [4, 5], 8: [4, 5], 9: [8], 10: [4, 5], 11:
```

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