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)
```

```
Certainly! Let's formulate the mathematical optimization model for the packing problem you described. We'll define the p

Parameters:
- I: Set of items to be packed, indexed by i
- B: Set of available boxes, indexed by j
- w[i]: Weight of item i
- C: Capacity of each box
- M: Upper bound on the number of boxes used

Decision Variables:
- x[i,j]: Binary variable, equals 1 if item i is packed into box j, 0 otherwise
- y[j]: Binary variable, equals 1 if box j is used, 0 otherwise

Objective Function:
- Minimize: sum(y[j] for j in B)
  - Explanation: The objective is to minimize the total number of boxes used.

Constraints:
1. Each item must be packed into exactly one box:
   - sum(x[i,j] for j in B) = 1, for all i in I
     - Explanation: This constraint ensures that each item is assigned to exactly one box.

2. The total weight of items packed into each box must not exceed the box capacity:
   - sum(w[i] * x[i,j] for i in I) <= C * y[j], for all j in B
     - Explanation: This constraint ensures that the total weight of items packed into each box does not exceed the box

3. The number of boxes used must not exceed the upper bound:
   - sum(y[j] for j in B) <= M
     - Explanation: This constraint ensures that the total number of boxes used does not exceed the specified upper boun

4. Binary restrictions on decision variables:
   - x[i,j] are binary variables, for all i in I and j in B
   - y[j] are binary variables, for all j in B
     - Explanation: These constraints specify that the decision variables x[i,j] and y[j] can only take binary values (0

The complete mathematical optimization model is as follows:

Minimize:
sum(y[j] for j in B)

Subject to:
1. sum(x[i,j] for j in B) = 1, for all i in I
2. sum(w[i] * x[i,j] for i in I) <= C * y[j], for all j in B
3. sum(y[j] for j in B) <= M
4. x[i,j] are binary variables, for all i in I and j in B
   y[j] are binary variables, for all j in B

This model aims to minimize the number of boxes used while ensuring that each item is packed into exactly one box, the t
```

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)
```

```
Great! Here's a Python code using Pyomo to solve the packing optimization problem you described:

```python
from pyomo.environ import *

# Sample data (you can replace this with your actual data)
items = ['item1', 'item2', 'item3', 'item4', 'item5']
boxes = ['box1', 'box2', 'box3']
weights = {'item1': 5, 'item2': 3, 'item3': 2, 'item4': 4, 'item5': 1}
capacity = 10
max_boxes = 2

# Create the Pyomo model
model = ConcreteModel()
```
```

```
    # Define the sets
    model.I = Set(initialize=items)
    model.B = Set(initialize=boxes)

    # Define the parameters
    model.w = Param(model.I, initialize=weights)
    model.C = Param(initialize=capacity)
    model.M = Param(initialize=max_boxes)

    # Define the decision variables
    model.x = Var(model.I, model.B, domain=Binary)
    model.y = Var(model.B, domain=Binary)

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

    # Define the constraints
    model.item_assignment = ConstraintList()
    for i in model.I:
        model.item_assignment.add(sum(model.x[i, j] for j in model.B) == 1)

    model.capacity_constraint = ConstraintList()
    for j in model.B:
        model.capacity_constraint.add(sum(model.w[i] * model.x[i, j] for i in model.I) <= model.C * model.y[j])

    model.max_boxes_constraint = Constraint(expr=sum(model.y[j] for j in model.B) <= model.M)

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

    # Print the results
    print("Optimization Status:", results.solver.status)
    print("Objective Value:", model.obj())

    print("\nItem Assignment:")
    for i in model.I:
        for j in model.B:
            if model.x[i, j]() == 1:
                print(f"Item {i} is packed into Box {j}")

    print("\nBox Usage:")
```

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 *

# Sample data (you can replace this with your actual data)
items = ['item1', 'item2', 'item3', 'item4', 'item5', 'item6', 'item7', 'item8', 'item9', 'item10', 'item11', 'item12', 'item
boxes = ['box1', 'box2', 'box3', 'box4', 'box5', 'box6', 'box7', 'box8', 'box9', 'box10', 'box11', 'box12', 'box13']
weights = [2, 2, 2, 2, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6, 7, 7, 8, 8]
capacity = 9
max_boxes = 13

# Create the Pyomo model
model = ConcreteModel()

# Define the sets
model.I = Set(initialize=items)
model.B = Set(initialize=boxes)

# Define the parameters
model.w = Param(model.I, initialize={items[i]: weights[i] for i in range(len(model.I))})
model.C = Param(initialize=capacity)
model.M = Param(initialize=max_boxes)

# Define the decision variables
model.x = Var(model.I, model.B, domain=Binary)
model.y = Var(model.B, domain=Binary)

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

# Define the constraints
```

```
# Define the constraints
model.item_assignment = ConstraintList()
for i in model.I:
    model.item_assignment.add(sum(model.x[i, j] for j in model.B) == 1)

model.capacity_constraint = ConstraintList()
for j in model.B:
    model.capacity_constraint.add(sum(model.w[i] * model.x[i, j] for i in model.I) <= model.C * model.y[j])

model.max_boxes_constraint = Constraint(expr=sum(model.y[j] for j in model.B) <= model.M)

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

# Print the results
print("Optimization Status:", results.solver.status)
print("Objective Value:", model.obj())

print("\nItem Assignment:")
for i in model.I:
    for j in model.B:
        if model.x[i, j]() == 1:
            print(f"Item {i} is packed into Box {j}")

print("\nBox Usage:")
for j in model.B:
    if model.y[j]() == 1:
        print(f"Box {j} is used")
```

```
Optimization Status: ok
Objective Value: 13.0

Item Assignment:
Item item1 is packed into Box box2
Item item2 is packed into Box box13
Item item3 is packed into Box box13
Item item4 is packed into Box box13
Item item5 is packed into Box box9
Item item6 is packed into Box box13
Item item7 is packed into Box box4
Item item8 is packed into Box box6
Item item9 is packed into Box box8
Item item10 is packed into Box box7
Item item11 is packed into Box box10
Item item12 is packed into Box box11
Item item13 is packed into Box box7
Item item14 is packed into Box box6
Item item15 is packed into Box box11
Item item16 is packed into Box box8
Item item17 is packed into Box box10
Item item18 is packed into Box box4
Item item19 is packed into Box box9
Item item20 is packed into Box box5
Item item21 is packed into Box box2
Item item22 is packed into Box box3
Item item23 is packed into Box box12
Item item24 is packed into Box box1

Box Usage:
Box box1 is used
Box box2 is used
Box box3 is used
Box box4 is used
Box box5 is used
Box box6 is used
Box box7 is used
Box box8 is used
Box box9 is used
Box box10 is used
Box box11 is used
Box box12 is used
Box box13 is used
```

```
bins = {'box1': [], 'box2': [], 'box3': [], 'box4': [], 'box5': [], 'box6': [], 'box7': [], 'box8': [], 'box9': [], 'box10':
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: {'box1': [8], 'box2': [2, 7], 'box3': [7], 'box4': [4, 5], 'box5': [6], 'box6': [4, 5], 'box7': [4, 5], 'b
```

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