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
```

```
Collecting anthropic
  Downloading anthropic-0.26.0-py3-none-any.whl (877 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 877.7/877.7 kB 4.3 MB/s eta 0:00:00
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-packages (from anthropic) (1.7.0)
Collecting httpx<1,>=0.23.0 (from anthropic)
  Downloading httpx-0.27.0-py3-none-any.whl (75 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 6.9 MB/s eta 0:00:00
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (2.7.1)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from anthropic) (1.3.1)
Requirement already satisfied: tokenizers>=0.13.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (0.19.1)
Requirement already satisfied: typing-extensions<5,>=4.7 in /usr/local/lib/python3.10/dist-packages (from anthropic) (4.
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->anthropic) (3
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->anthropi
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->anthropic) (20
Collecting httpcore==1.* (from httpx<1,>=0.23.0->anthropic)
  Downloading httpcore-1.0.5-py3-none-any.whl (77 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 8.9 MB/s eta 0:00:00
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->anthropic)
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 5.8 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.
Requirement already satisfied: pydantic-core==2.18.2 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.0
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/lib/python3.10/dist-packages (from tokenizers>
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->t
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->t
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4
```

```
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->huggi
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface
Installing collected packages: h11, httpcore, httpx, anthropic
Successfully installed anthropic-0.26.0 h11-0.14.0 httpcore-1.0.5 httpx-0.27.0
```

```
# 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.

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_ij: Binary variable indicating whether item i is packed into box j (1 if packed, 0 otherwise)
- y_j: Binary variable indicating whether box j is used (1 if used, 0 otherwise)

Objective Function:
- Minimize the total number of boxes used:
  min Σ_j y_j

Constraints:
1. Each item must be packed into exactly one box:
   Σ_j x_ij = 1, ∀i ∈ I

2. The total weight of items packed into each box must not exceed the box capacity:
   Σ_i w_i * x_ij ≤ C * y_j, ∀j ∈ B

3. The number of boxes used must not exceed the upper bound:
   Σ_j y_j ≤ M

4. Binary restrictions on the decision variables:
   x_ij ∈ {0, 1}, ∀i ∈ I, ∀j ∈ B
   y_j ∈ {0, 1}, ∀j ∈ B

The complete mathematical optimization model is as follows:

min Σ_j y_j

subject to:
1. Σ_j x_ij = 1, ∀i ∈ I
2. Σ_i w_i * x_ij ≤ C * y_j, ∀j ∈ B
3. Σ_j y_j ≤ M
4. x_ij ∈ {0, 1}, ∀i ∈ I, ∀j ∈ B
   y_j ∈ {0, 1}, ∀j ∈ B

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)
```

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

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

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

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

# Define objective function
def obj_rule(model):
    return sum(model.y[j] for j in model.B)
model.obj = Objective(rule=obj_rule, sense=minimize)

# Define constraints
def item_packing_rule(model, i):
    return sum(model.x[i, j] for j in model.B) == 1
model.item_packing = Constraint(model.I, rule=item_packing_rule)

def capacity_rule(model, j):
    return sum(model.w[i] * model.x[i, j] for i in model.I) <= model.C * model.y[j]
model.capacity = Constraint(model.B, rule=capacity_rule)

def box_limit_rule(model):
    return sum(model.y[j] for j in model.B) <= model.M
model.box_limit = Constraint(rule=box_limit_rule)

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

# Print the results
print("Optimal solution found:")
print("Objective value:", value(model.obj))

for j in model.B:
    if value(model.y[j]) == 1:
        print(f"Box {j} is used.")
        for i in model.I:
            if value(model.x[i, j]) == 1:
                print(f"  Item {i} is packed into box {j}.")
```

In this code, we first define the sample data for items, boxes, weights, capacity, and the maximum number of boxes all

We then create a Pyomo model and define the sets, parameters, decision variables, objective function, and constraints

The objective function minimizes the total number of boxes used. The constraints ensure that each item is packed into

Finally, we solve the model using the GLPK solver (you may need to install it separately) and print the optimal soluti

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 (replace 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, 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 sets
model.I = Set(initialize=items)
model.B = Set(initialize=boxes)

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

# Define objective function
def obj_rule(model):
    return sum(model.y[j] for j in model.B)
model.obj = Objective(rule=obj_rule, sense=minimize)

# Define constraints
def item_packing_rule(model, i):
    return sum(model.x[i, j] for j in model.B) == 1
model.item_packing = Constraint(model.I, rule=item_packing_rule)

def capacity_rule(model, j):
    return sum(model.w[i] * model.x[i, j] for i in model.I) <= model.C * model.y[j]
model.capacity = Constraint(model.B, rule=capacity_rule)

def box_limit_rule(model):
    return sum(model.y[j] for j in model.B) <= model.M
model.box_limit = Constraint(rule=box_limit_rule)

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

# Print the results
print("Optimal solution found:")
print("Objective value:", value(model.obj))

for j in model.B:
    if value(model.y[j]) == 1:
        print(f"Box {j} is used.")
        for i in model.I:
            if value(model.x[i, j]) == 1:
                print(f"  Item {i} is packed into box {j}.")
```

```
Optimal solution found:
Objective value: 13.0
Box box1 is used.
  Item item24 is packed into box box1.
Box box2 is used.
  Item item1 is packed into box box2.
  Item item21 is packed into box box2.
Box box3 is used.
  Item item22 is packed into box box3.
Box box4 is used.
  Item item7 is packed into box box4.
  Item item18 is packed into box box4.
Box box5 is used.
  Item item20 is packed into box box5.
Box box6 is used.
  Item item8 is packed into box box6.
  Item item14 is packed into box box6.
Box box7 is used.
  Item item10 is packed into box box7.
  Item item13 is packed into box box7.
Box box8 is used.
  Item item9 is packed into box box8.
  Item item16 is packed into box box8.
Box box9 is used.
  Item item5 is packed into box box9.
  Item item19 is packed into box box9.
Box box10 is used.
```

```
    Item item11 is packed into box box10.
    Item item17 is packed into box box10.
  Box box11 is used.
    Item item12 is packed into box box11.
    Item item15 is packed into box box11.
  Box box12 is used.
    Item item23 is packed into box box12.
  Box box13 is used.
    Item item2 is packed into box box13.
    Item item3 is packed into box box13.
    Item item4 is packed into box box13.
    Item item6 is packed into box box13.
```

```
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)