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/NL/NL3.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:
 A buyer needs to acquire 239,600,480 units of a product and is considering bids from five suppliers, labeled A through
Each vendor has proposed different pricing structures, incorporating both setup fees and variable unit costs that change

The buyer's objective is to allocate the order among these suppliers to minimize overall costs, accounting for both setu

Vendor A offers a set up cost of $3855.34 and a unit cost of $61.150 per thousand of units.
Vendor A can supply up to 33 million units.

Vendor B offers a set up cost of $125,804.84 if purchasing between 22,000,000-70,000,000 units from vendor B with a unit
If purchasing between 70,000,001-100,000,000 units from vendor B, the set up cost increases to $269304.84 and the unit c
If purchasing between 100,000,001-150,000,000 units from vendor B, the unit cost per thousand units further decreases to
If purchasing between 150,000,001 and 160,000,000 units from vendor B, the unit cost is $62.119 per thousand units and t

Vendor C offers set up costs of $13,456.00 and a unit cost of $62.019 per thousand units.
Vendor C can supply up to 165.6 million units. Vendor D offers set up costs of $6,583.98 and a unit cost of $72.488 for

Vendor D can supply up to 12 million units at a price of $72.488 per thousand units and with a set up cost of $6583.98.

Vendor E offers free set up if purchasing between 0 and 42 million units of vendor E with a unit price of $70.150 per th
If purchasing between 42,000,001 and 77 million units from vendor E, the unit cost starts at $68.150 per thousand units,

Note that zero units may be purchased from vendor B: otherwise no positive number of units less than 22,000,000 may be p
```

```
!pip install anthropic
```

```
Collecting anthropic
  Downloading anthropic-0.26.0-py3-none-any.whl (877 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 877.7/877.7 kB 5.8 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 5.5 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 6.7 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)
```

```
Here's a mathematical optimization model for the given problem:

Parameters:
- Q: Total quantity required (239,600,480 units)
- SA: Setup cost for vendor A ($3,855.34)
- SB1, SB2, SB3, SB4: Setup costs for vendor B based on quantity ranges
- SC: Setup cost for vendor C ($13,456.00)
- SD: Setup cost for vendor D ($6,583.98)
- SE1, SE2: Setup costs for vendor E based on quantity ranges
- UA: Unit cost per thousand units for vendor A ($61.150)
- UB1, UB2, UB3, UB4: Unit costs per thousand units for vendor B based on quantity ranges
- UC: Unit cost per thousand units for vendor C ($62.019)
- UD: Unit cost per thousand units for vendor D ($72.488)
- UE1, UE2: Unit costs per thousand units for vendor E based on quantity ranges
- MA, MC, MD: Maximum quantities that can be supplied by vendors A, C, and D
- MB1, MB2, MB3, MB4: Maximum quantities for each range of vendor B
- ME1, ME2: Maximum quantities for each range of vendor E

Decision Variables:
- xA: Quantity purchased from vendor A
- xB1, xB2, xB3, xB4: Quantities purchased from vendor B for each range
- xC: Quantity purchased from vendor C
- xD: Quantity purchased from vendor D
- xE1, xE2: Quantities purchased from vendor E for each range
- yB1, yB2, yB3, yB4: Binary variables indicating the selected range for vendor B
- yE1, yE2: Binary variables indicating the selected range for vendor E

Objective Function:
Minimize: SA + UA * xA + SB1 * yB1 + SB2 * yB2 + SB3 * yB3 + SB4 * yB4 + UB1 * xB1 + UB2 * xB2 + UB3 * xB3 + UB4 * xB4 +

Constraints:
1. Total quantity constraint: xA + xB1 + xB2 + xB3 + xB4 + xC + xD + xE1 + xE2 = Q
```

```
2. Capacity constraints:
   - xA ≤ MA
   - xB1 ≤ MB1 * yB1
   - xB2 ≤ MB2 * yB2
   - xB3 ≤ MB3 * yB3
   - xB4 ≤ MB4 * yB4
   - xC ≤ MC
   - xD ≤ MD
   - xE1 ≤ ME1 * yE1
   - xE2 ≤ ME2 * yE2
3. Range selection constraints for vendor B:
   - yB1 + yB2 + yB3 + yB4 ≤ 1
   - 22000000 * yB1 ≤ xB1 ≤ 70000000 * yB1
   - 70000001 * yB2 ≤ xB2 ≤ 100000000 * yB2
   - 100000001 * yB3 ≤ xB3 ≤ 150000000 * yB3
   - 150000001 * yB4 ≤ xB4 ≤ 160000000 * yB4
4. Range selection constraints for vendor E:
   - yE1 + yE2 = 1
   - xE1 ≤ 42000000 * yE1
   - 42000001 * yE2 ≤ xE2 ≤ 77000000 * yE2
5. Non-negativity constraints: xA, xB1, xB2, xB3, xB4, xC, xD, xE1, xE2 ≥ 0
6. Binary constraints: yB1, yB2, yB3, yB4, yE1, yE2 ∈ {0, 1}
```

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)
```

```
            0.0005 * (model.xE[2] / 1000 - 42000) * model.xE[2]
        )
```

```
    print("Quantity purchased from vendor B:")
    for i in range(1, 5):
        print(f"  Range {i}: {value(model.xB[i])}")
    print("Quantity purchased from vendor C:", value(model.xC))
    print("Quantity purchased from vendor D:", value(model.xD))
    print("Quantity purchased from vendor E:")
    for i in range(1, 3):
        print(f"  Range {i}: {value(model.xE[i])}")
    ```

    This code defines the optimization model using Pyomo, including the parameters, decision variables, objective function

    The objective function is defined using the `objective_rule` function, which calculates the total cost based on the se
```

## 4. Input Problem Data and Test Model Code

```
%%capture
import sys
import os

if 'google.colab' in sys.modules:
    !pip install idaes-pse --pre
    !idaes get-extensions --to ./bin
    os.environ['PATH'] += ':bin'
```

```
from pyomo.environ import *

# Define the model
model = ConcreteModel()

# Define the parameters
model.Q = 239600480  # Total quantity required
model.SA = 3855.34  # Setup cost for vendor A
model.SB = [125804.84, 269304.84, 464304.84, 761304.84]  # Setup costs for vendor B based on quantity ranges
model.SC = 13456.00  # Setup cost for vendor C
model.SD = 6583.98  # Setup cost for vendor D
model.SE = [0, 0]  # Setup costs for vendor E based on quantity ranges
model.UA = 61.150  # Unit cost per thousand units for vendor A
model.UB = [68.099, 66.049, 64.099, 62.119]  # Unit costs per thousand units for vendor B based on quantity ranges
model.UC = 62.019  # Unit cost per thousand units for vendor C
model.UD = 72.488  # Unit cost per thousand units for vendor D
model.UE = [70.150, 68.150]  # Unit costs per thousand units for vendor E based on quantity ranges
model.MA = 70000000  # Maximum quantity that can be supplied by vendor A
model.MB = [70000000, 100000000, 150000000, 160000000]  # Maximum quantities for each range of vendor B
model.MC = 75000000  # Maximum quantity that can be supplied by vendor C
model.MD = 45000000  # Maximum quantity that can be supplied by vendor D
model.ME = [42000000, 77000000]  # Maximum quantities for each range of vendor E

# Define the decision variables
model.xA = Var(domain=NonNegativeReals)  # Quantity purchased from vendor A
model.xB = Var(RangeSet(1, 4), domain=NonNegativeReals)  # Quantities purchased from vendor B for each range
model.xC = Var(domain=NonNegativeReals)  # Quantity purchased from vendor C
model.xD = Var(domain=NonNegativeReals)  # Quantity purchased from vendor D
model.xE = Var(RangeSet(1, 2), domain=NonNegativeReals)  # Quantities purchased from vendor E for each range
model.yB = Var(RangeSet(1, 4), domain=Binary)  # Binary variables indicating the selected range for vendor B
model.yE = Var(RangeSet(1, 2), domain=Binary)  # Binary variables indicating the selected range for vendor E

# Define the objective function
def objective_rule(model):
    return (
        model.SA + model.UA * model.xA +
        sum(model.SB[i-1] * model.yB[i] + model.UB[i-1] * model.xB[i] for i in range(1, 5)) +
        model.SC + model.UC * model.xC +
        model.SD + model.UD * model.xD +
        sum(model.SE[i-1] * model.yE[i] + model.UE[i-1] * model.xE[i] for i in range(1, 3)) -
        0.0005 * (model.xE[2] / 1000 - 42000) * model.xE[2]
    )
model.objective = Objective(rule=objective_rule, sense=minimize)

# Define the constraints
def total_quantity_constraint(model):
    return model.xA + sum(model.xB[i] for i in range(1, 5)) + model.xC + model.xD + sum(model.xE[i] for i in range(1, 3)) ==
model.total_quantity = Constraint(rule=total_quantity_constraint)

def capacity_constraints(model):
    yield model.xA <= model.MA
    for i in range(1, 5):
        yield model.xB[i] <= model.MB[i-1] * model.yB[i]
    yield model.xC <= model.MC
    yield model.xD <= model.MD
    for i in range(1, 3):
```
```

```
    for i in range(1, 3):
        yield model.xE[i] <= model.ME[i-1] * model.yE[i]
model.capacity = ConstraintList(rule=capacity_constraints)

def vendor_B_range_selection(model):
    yield sum(model.yB[i] for i in range(1, 5)) <= 1
    yield model.xB[1] >= 22000000 * model.yB[1]
    yield model.xB[1] <= 70000000 * model.yB[1]
    yield model.xB[2] >= 70000001 * model.yB[2]
    yield model.xB[2] <= 100000000 * model.yB[2]
    yield model.xB[3] >= 100000001 * model.yB[3]
    yield model.xB[3] <= 150000000 * model.yB[3]
    yield model.xB[4] >= 150000001 * model.yB[4]
    yield model.xB[4] <= 160000000 * model.yB[4]
model.vendor_B_range = ConstraintList(rule=vendor_B_range_selection)

def vendor_E_range_selection(model):
    yield model.yE[1] + model.yE[2] == 1
    yield model.xE[1] <= 42000000 * model.yE[1]
    yield model.xE[2] >= 42000001 * model.yE[2]
    yield model.xE[2] <= 77000000 * model.yE[2]
model.vendor_E_range = ConstraintList(rule=vendor_E_range_selection)

# Solve the model
opt = SolverFactory('couenne')
results = opt.solve(model)

# Print the results
print("Objective value:", value(model.objective))
print("Quantity purchased from vendor A:", value(model.xA))
print("Quantity purchased from vendor B:")
for i in range(1, 5):
    print(f"  Range {i}: {value(model.xB[i])}")
print("Quantity purchased from vendor C:", value(model.xC))
print("Quantity purchased from vendor D:", value(model.xD))
print("Quantity purchased from vendor E:")
for i in range(1, 3):
    print(f"  Range {i}: {value(model.xE[i])}")
```

```
Objective value: 13989204553.129002
Quantity purchased from vendor A: 12600478.99999953
Quantity purchased from vendor B:
  Range 1: 0.0
  Range 2: 0.0
  Range 3: 0.0
  Range 4: 150000001.00000048
Quantity purchased from vendor C: 0.0
Quantity purchased from vendor D: 0.0
Quantity purchased from vendor E:
  Range 1: 0.0
  Range 2: 77000000.0
```

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