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
# Variables Prompt
prompt11_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt11_MathematicalModel.txt'

# Objective Prompt
prompt12_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt12_MathematicalModel.txt'

# Constraint Prompt
prompt13_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt13_MathematicalModel.txt'

# Code Prompt
prompt2_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt2_PyomoCode.txt'
problem_desc_path = '/content/drive/MyDrive/Thesis/ProblemDescriptions/NL/NL3.txt'

prompt11_file = open(prompt11_path, "r")
prompt12_file = open(prompt12_path, "r")
prompt13_file = open(prompt13_path, "r")
prompt2_file = open(prompt2_path, "r")
problem_desc_file = open(problem_desc_path, "r")

prompt11 = prompt11_file.read()
print("Prompt 1.1 (Variables):\n", prompt11)

prompt12 = prompt12_file.read()
print("Prompt 1.2 (Objctive):\n", prompt12)

prompt13 = prompt13_file.read()
print("Prompt 1.3 (Constraints):\n", prompt13)

prompt2 = prompt2_file.read()
print("Prompt 2:\n", prompt2)

problem_desc = problem_desc_file.read()
print("Problem Description:\n", problem_desc)
```

```
Prompt 1.1 (Variables):
 Please formulate only the variables for this mathematical optimization problem.
Prompt 1.2 (Objctive):
 Please formulate only the objective function for this mathematical optimization problem.
Prompt 1.3 (Constraints):
 Please formulate only the constraints for this mathematical optimization problem.
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
```

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 1.1: Create Variables for Mathematical Model

```
message11 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt11,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)

response_p11 = message11.content[0].text


# Print response
print(response_p11)
```

```
Here are the variables for this optimization problem:

Let x_A be the number of units (in thousands) purchased from vendor A.
0 ≤ x_A ≤ 33,000

Let x_B1, x_B2, x_B3, x_B4 be the number of units (in thousands) purchased from vendor B in each of the 4 possible quant
22,000 ≤ x_B1 ≤ 70,000
70,000 < x_B2 ≤ 100,000
100,000 < x_B3 ≤ 150,000
150,000 < x_B4 ≤ 160,000

Let x_C be the number of units (in thousands) purchased from vendor C.
0 ≤ x_C ≤ 165,600

Let x_D be the number of units (in thousands) purchased from vendor D.
0 ≤ x_D ≤ 12,000

Let x_E1 be the number of units (in thousands) purchased from vendor E between 0 and 42,000.
0 ≤ x_E1 ≤ 42,000

Let x_E2 be the number of units (in thousands) purchased from vendor E between 42,000 and 77,000. The unit price depends
0 ≤ x_E2 ≤ 35,000

The total quantity purchased must equal the buyer's requirement:
x_A + x_B1 + x_B2 + x_B3 + x_B4 + x_C + x_D + x_E1 + x_E2 = 239,600.480
```

1. Prompt 1.2: Create Objective for Mathematical Model

```
message12 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt12,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11)}
    ]
)

response_p12 = message12.content[0].text


# Print response
print(response_p12)
```

```
Here is the objective function to minimize the total cost:

Minimize
3855.34 + 61.150*x_A +
125804.84*y_B1 + 68.099*x_B1 +
269304.84*y_B2 + 66.049*x_B2 +
464304.84*y_B3 + 64.099*x_B3 +
761304.84*y_B4 + 62.119*x_B4 +
13456 + 62.019*x_C +
6583.98 + 72.488*x_D +
70.150*x_E1 +
84000*y_E2 + 68.150*x_E2 - 0.0006815*x_E2^2

Where:
- y_B1, y_B2, y_B3, y_B4 are binary variables, with y_Bi = 1 if x_Bi > 0, and 0 otherwise. Only one of y_B1, y_B2, y_B3,
- y_E2 is a binary variable, with y_E2 = 1 if x_E2 > 0, and 0 otherwise.

The quadratic term -0.0006815*x_E2^2 captures the price decrease of 0.05% per 1000 units for vendor E in the range 42,00
```

1. Prompt 1.3: Create Constraints for Mathematical Model

```
message13 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt13,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11 + response_p12)}
    ]
)

response_p13 = message13.content[0].text


# Print response
print(response_p13)
```

```
Here are the constraints for this optimization problem:

1. Quantity constraints:
   0 ≤ x_A ≤ 33,000
   22,000*y_B1 ≤ x_B1 ≤ 70,000*y_B1
   70,000*y_B2 < x_B2 ≤ 100,000*y_B2
   100,000*y_B3 < x_B3 ≤ 150,000*y_B3
   150,000*y_B4 < x_B4 ≤ 160,000*y_B4
   0 ≤ x_C ≤ 165,600
   0 ≤ x_D ≤ 12,000
   0 ≤ x_E1 ≤ 42,000
   0 ≤ x_E2 ≤ 35,000

2. Binary variable constraints for vendor B:
   y_B1 + y_B2 + y_B3 + y_B4 ≤ 1
   y_B1, y_B2, y_B3, y_B4 are binary

3. Binary variable constraint for vendor E:
   y_E2 is binary

4. Total quantity constraint:
   x_A + x_B1 + x_B2 + x_B3 + x_B4 + x_C + x_D + x_E1 + x_E2 = 239,600.480

5. Non-negativity constraints:
   x_A, x_B1, x_B2, x_B3, x_B4, x_C, x_D, x_E1, x_E2 ≥ 0

Note: The binary variables y_B1, y_B2, y_B3, y_B4 ensure that only one of the quantity ranges for vendor B can be select
```

2. Prompt 2: Write Code based on Mathematical Model

In case more than the component of each step was generated, use "last instance"/ "newest version" of the components.

Example: Prompt 1.1 (Variables): Model generates only variables Prompt 1.2 (Objectvie): Model generates objective and constraints Prompt 1.3 (Constraints): Model generates only constraints

Use Variables from step 1.1, use objective from 1.2 and use constraints from 1.3 (since constraints version 1.3 is more recent than constraints version 1.2).

This set up allows for potential self-correction of the model throughout the three prompts.

Correctness of each component is still generated in the respective step, so self-correction is independent of this.

```
message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": (response_p11+ response_p12 + response_p13)}
    ]
)

response_p2 = message2.content[0].text


# Print response
print(response_p2)
```

```
Here's the Python code using Pyomo to solve the optimization problem:

```python
from pyomo.environ import *

# Create a concrete model
model = ConcreteModel()

# Define decision variables
model.x_A = Var(bounds=(0, 33000))
model.x_B1 = Var(bounds=(0, 70000))
model.x_B2 = Var(bounds=(0, 100000))
model.x_B3 = Var(bounds=(0, 150000))
model.x_B4 = Var(bounds=(0, 160000))
model.x_C = Var(bounds=(0, 165600))
model.x_D = Var(bounds=(0, 12000))
model.x_E1 = Var(bounds=(0, 42000))
model.x_E2 = Var(bounds=(0, 35000))

model.y_B1 = Var(domain=Binary)
model.y_B2 = Var(domain=Binary)
model.y_B3 = Var(domain=Binary)
model.y_B4 = Var(domain=Binary)
model.y_E2 = Var(domain=Binary)

# Define objective function
def objective_rule(model):
    return (
        3855.34 + 61.150 * model.x_A +
        125804.84 * model.y_B1 + 68.099 * model.x_B1 +
        269304.84 * model.y_B2 + 66.049 * model.x_B2 +
        464304.84 * model.y_B3 + 64.099 * model.x_B3 +
        761304.84 * model.y_B4 + 62.119 * model.x_B4 +
        13456 + 62.019 * model.x_C +
        6583.98 + 72.488 * model.x_D +
        70.150 * model.x_E1 +
        84000 * model.y_E2 + 68.150 * model.x_E2 - 0.0006815 * model.x_E2 ** 2
    )
model.objective = Objective(rule=objective_rule, sense=minimize)

# Define constraints
model.quantity_constraint_B1 = Constraint(expr=model.x_B1 >= 22000 * model.y_B1)
model.quantity_constraint_B2 = Constraint(expr=model.x_B2 <= 100000 * model.y_B2)
model.quantity_constraint_B2_lower = Constraint(expr=model.x_B2 >= 70000 * model.y_B2 + 1)
model.quantity_constraint_B3 = Constraint(expr=model.x_B3 <= 150000 * model.y_B3)
model.quantity_constraint_B3_lower = Constraint(expr=model.x_B3 >= 100000 * model.y_B3 + 1)
model.quantity_constraint_B4 = Constraint(expr=model.x_B4 <= 160000 * model.y_B4)
model.quantity_constraint_B4_lower = Constraint(expr=model.x_B4 >= 150000 * model.y_B4 + 1)

model.binary_constraint_B = Constraint(expr=model.y_B1 + model.y_B2 + model.y_B3 + model.y_B4 <= 1)

model.total_quantity_constraint = Constraint(expr=model.x_A + model.x_B1 + model.x_B2 + model.x_B3 + model.x_B4 + mode

# Solve the optimization problem
solver = SolverFactory('gurobi')  # Specify the solver (e.g., 'gurobi', 'cplex', 'glpk')
results = solver.solve(model)
```
```

4. Input Problem Data and Test Model Code

```
# Download Gurobi
!wget https://packages.gurobi.com/9.5/gurobi9.5.2_linux64.tar.gz

# Extract the tarball
!tar -xvzf gurobi9.5.2_linux64.tar.gz

# Set up environment variables for Gurobi
import os
os.environ['GUROBI_HOME'] = "/content/gurobi952/linux64"
os.environ['PATH'] += ":/content/gurobi952/linux64/bin"
os.environ['LD_LIBRARY_PATH'] = "/content/gurobi952/linux64/lib"
```

Show hidden output

```
import shutil
shutil.move('/content/drive/MyDrive/gurobi.lic', '/root/gurobi.lic')
```

```
'/root/gurobi.lic'
```

```
# Installing pyomo nd solver
!pip install -q pyomo
!pip install pandas
```

Show hidden output

```
from pyomo.environ import *

# Create a concrete model
model = ConcreteModel()

# Define decision variables
model.x_A = Var(bounds=(0, 33000))
model.x_B1 = Var(bounds=(0, 70000))
model.x_B2 = Var(bounds=(0, 100000))
model.x_B3 = Var(bounds=(0, 150000))
model.x_B4 = Var(bounds=(0, 160000))
model.x_C = Var(bounds=(0, 165600))
model.x_D = Var(bounds=(0, 12000))
model.x_E1 = Var(bounds=(0, 42000))
model.x_E2 = Var(bounds=(0, 35000))

model.y_B1 = Var(domain=Binary)
model.y_B2 = Var(domain=Binary)
model.y_B3 = Var(domain=Binary)
model.y_B4 = Var(domain=Binary)
model.y_E2 = Var(domain=Binary)

# Define objective function
def objective_rule(model):
    return (
        3855.34 + 61.150 * model.x_A +
        125804.84 * model.y_B1 + 68.099 * model.x_B1 +
        269304.84 * model.y_B2 + 66.049 * model.x_B2 +
        464304.84 * model.y_B3 + 64.099 * model.x_B3 +
        761304.84 * model.y_B4 + 62.119 * model.x_B4 +
        13456 + 62.019 * model.x_C +
        6583.98 + 72.488 * model.x_D +
        70.150 * model.x_E1 +
        84000 * model.y_E2 + 68.150 * model.x_E2 - 0.0006815 * model.x_E2 ** 2
    )
model.objective = Objective(rule=objective_rule, sense=minimize)

# Define constraints
model.quantity_constraint_B1 = Constraint(expr=model.x_B1 >= 22000 * model.y_B1)
model.quantity_constraint_B2 = Constraint(expr=model.x_B2 <= 100000 * model.y_B2)
model.quantity_constraint_B2_lower = Constraint(expr=model.x_B2 >= 70000 * model.y_B2 + 1)
model.quantity_constraint_B3 = Constraint(expr=model.x_B3 <= 150000 * model.y_B3)
model.quantity_constraint_B3_lower = Constraint(expr=model.x_B3 >= 100000 * model.y_B3 + 1)
model.quantity_constraint_B4 = Constraint(expr=model.x_B4 <= 160000 * model.y_B4)
model.quantity_constraint_B4_lower = Constraint(expr=model.x_B4 >= 150000 * model.y_B4 + 1)

model.binary_constraint_B = Constraint(expr=model.y_B1 + model.y_B2 + model.y_B3 + model.y_B4 <= 1)

model.total_quantity_constraint = Constraint(expr=model.x_A + model.x_B1 + model.x_B2 + model.x_B3 + model.x_B4 + model.x_C +
```

```
model.total_quantity_constraint = Constraint(expr=model.x_A + model.x_B1 + model.x_B2 + model.x_B3 + model.x_B4 + model.x_C +

# Solve the optimization problem
solver = SolverFactory('gurobi')  # Specify the solver (e.g., 'gurobi', 'cplex', 'glpk')
results = solver.solve(model)

# Print the results
print("Objective value:", model.objective())
print("x_A:", model.x_A())
print("x_B1:", model.x_B1())
print("x_B2:", model.x_B2())
print("x_B3:", model.x_B3())
print("x_B4:", model.x_B4())
print("x_C:", model.x_C())
print("x_D:", model.x_D())
print("x_E1:", model.x_E1())
print("x_E2:", model.x_E2())
```

```
WARNING:pyomo.core:Loading a SolverResults object with a warning status into model.name="unknown";
  - termination condition: infeasible
  - message from solver: Model was proven to be infeasible.
ERROR:pyomo.common.numeric_types:evaluating object as numeric value: x_A
    (object: <class 'pyomo.core.base.var.ScalarVar'>)
No value for uninitialized NumericValue object x_A
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-3f7f63829fd9> in <cell line: 56>()
     54
     55 # Print the results
---> 56 print("Objective value:", model.objective())
     57 print("x_A:", model.x_A())
     58 print("x_B1:", model.x_B1())

                                  6 frames
/usr/local/lib/python3.10/dist-packages/pyomo/common/numeric_types.py in value(obj, exception)
    382             tmp = obj(exception=True)
    383             if tmp is None:
--> 384                 raise ValueError(
    385                     "No value for uninitialized NumericValue object %s" % (obj.name,)
    386                 )

ValueError: No value for uninitialized NumericValue object x_A
```

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