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/IP/IP2.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:
 Your goal is to invest in several of 10 possible investment strategies in the most optimal way. The historic returns of

The costs for investing in a given investment is stored in a vector A, which has one value for each strategy in order.
The values are: [80, 340, 410, 50, 180, 221, 15, 348, 191, 225]

You can only invest once into an investment.

Unfortunately due to other costs and inflation, your available budget at this time is uncertain. There are four possible
The tolerable probability of exceeding the budget is 0.4.

Please formulate a mean-variance mathematical model for this optimization problem, considering the past performance of i
```

```
!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)
```

```
Let's define the variables for this mean-variance optimization problem:

Decision variables:
x_i: binary variable indicating whether to invest in strategy i (x_i = 1) or not (x_i = 0), for i = 1, 2, ..., 10.

Parameters:
r_ij: historical return of strategy i in period j, for i = 1, 2, ..., 10 and j = 1, 2, ..., n, where n is the number of
A_i: cost of investing in strategy i, for i = 1, 2, ..., 10.
B_k: available budget in scenario k, for k = 1, 2, 3, 4.
p_k: probability of scenario k occurring, for k = 1, 2, 3, 4.
α: tolerable probability of exceeding the budget (α = 0.4).
r: risk parameter (r = 2).

Auxiliary variables:
μ_i: mean return of strategy i, calculated as μ_i = (1/n) * Σ_j r_ij, for i = 1, 2, ..., 10.
σ_ij: covariance between the returns of strategies i and j, calculated as σ_ij = (1/(n-1)) * Σ_k (r_ik - μ_i) * (r_jk -
y_k: binary variable indicating whether the budget is exceeded in scenario k (y_k = 1) or not (y_k = 0), for k = 1, 2, 3

These are the variables needed to formulate the mean-variance optimization problem with uncertain budget constraints.
```

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)
```

```
Objective function:
Maximize: Σ_i μ_i * x_i - r * Σ_i Σ_j σ_ij * x_i * x_j

Constraints:
1. Budget constraint for each scenario k:
   Σ_i A_i * x_i ≤ B_k + M * y_k, for k = 1, 2, 3, 4.
   (M is a large constant, allowing the budget to be exceeded when y_k = 1)

2. Probability of exceeding the budget:
   Σ_k p_k * y_k ≤ α
```

```
3. Binary constraints:
   x_i ∈ {0, 1}, for i = 1, 2, ..., 10.
   y_k ∈ {0, 1}, for k = 1, 2, 3, 4.

The objective function maximizes the expected return (Σ_i μ_i * x_i) while penalizing the risk, represented by the varia

The first set of constraints ensures that the total cost of the selected strategies does not exceed the available budget

The second constraint limits the probability of exceeding the budget to the tolerable level α.

Finally, the binary constraints ensure that the decision variables x_i and y_k can only take values of 0 or 1.
```

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 the mean-variance optimization problem with uncertain budget:

1. Budget constraint for each scenario k:
   Σ_i A_i * x_i ≤ B_1 + M * y_1, for k = 1.
   Σ_i A_i * x_i ≤ B_2 + M * y_2, for k = 2.
   Σ_i A_i * x_i ≤ B_3 + M * y_3, for k = 3.
   Σ_i A_i * x_i ≤ B_4 + M * y_4, for k = 4.
   (M is a large constant, allowing the budget to be exceeded when y_k = 1)

2. Probability of exceeding the budget:
   0.55 * y_1 + 0.4 * y_2 + 0.04 * y_3 + 0.01 * y_4 ≤ 0.4

3. Binary constraints:
   x_i ∈ {0, 1}, for i = 1, 2, ..., 10.
   y_k ∈ {0, 1}, for k = 1, 2, 3, 4.

4. Investment constraint:
   Σ_i x_i ≤ 1, for i = 1, 2, ..., 10.

The budget constraints ensure that the total cost of the selected strategies does not exceed the available budget in eac

The probability constraint limits the probability of exceeding the budget to the tolerable level of 0.4, considering the

The binary constraints ensure that the decision variables x_i and y_k can only take values of 0 or 1.

The investment constraint ensures that you can only invest once into an investment strategy.
```

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.

```
# Extract objective related answer part from response two
response_p12_adjusted = """
Objective function:
Maximize: Σ_i μ_i * x_i - r * Σ_i Σ_j σ_ij * x_i * x_j

The objective function maximizes the expected return (Σ_i μ_i * x_i) while penalizing the risk, represented by the variance
"""



message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": (response_p11 + response_p12_adjusted + response_p13)}
    ]
)

response_p2 = message2.content[0].text


# Print response
print(response_p2)
```

```
Here's a Python code using Pyomo to solve the mean-variance optimization problem with uncertain budget constraints:

```python
import pyomo.environ as pyo
import numpy as np

# Sample data (replace with your actual data)
n = 10  # Number of historical periods
m = 10  # Number of strategies
scenarios = 4  # Number of budget scenarios

# Generate sample returns (replace with your actual returns)
returns = np.random.rand(m, n)

# Generate sample costs for each strategy (replace with your actual costs)
costs = np.random.randint(10, 50, m)

# Sample budgets for each scenario (replace with your actual budgets)
budgets = [100, 120, 80, 90]

# Sample probabilities for each scenario (replace with your actual probabilities)
probabilities = [0.55, 0.4, 0.04, 0.01]

# Create the Pyomo model
model = pyo.ConcreteModel()

# Define sets
model.strategies = pyo.RangeSet(1, m)
model.scenarios = pyo.RangeSet(1, scenarios)

# Define decision variables
model.x = pyo.Var(model.strategies, domain=pyo.Binary)
model.y = pyo.Var(model.scenarios, domain=pyo.Binary)

# Define parameters
model.returns = pyo.Param(model.strategies, initialize=lambda model, i: returns[i-1].mean())
model.costs = pyo.Param(model.strategies, initialize=lambda model, i: costs[i-1])
model.budgets = pyo.Param(model.scenarios, initialize=lambda model, k: budgets[k-1])
model.probabilities = pyo.Param(model.scenarios, initialize=lambda model, k: probabilities[k-1])

# Calculate covariance matrix
covariance_matrix = np.cov(returns)

# Define objective function
def objective_rule(model):
    return sum(model.returns[i] * model.x[i] for i in model.strategies) - \
           2 * sum(covariance_matrix[i-1, j-1] * model.x[i] * model.x[j] for i in model.strategies for j in model.st

model.objective = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

# Define budget constraints for each scenario
def budget_constraint_rule(model, k):
    return sum(model.costs[i] * model.x[i] for i in model.strategies) <= model.budgets[k] + 1e6 * model.y[k]

model.budget_constraint = pyo.Constraint(model.scenarios, rule=budget_constraint_rule)

# Define probability of exceeding the budget constraint
```

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

```
import pyomo.environ as pyo
import numpy as np
import pandas as pd

# Sample data (replace with your actual data)
scenarios = 4  # Number of budget scenarios

# Generate sample returns (replace with your actual returns)
data = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/IP/IP2/investments_data.csv", header=None, sep=';')
returns = data.to_numpy()
n, m = returns.shape

# Generate sample costs for each strategy (replace with your actual costs)
costs = [80, 340, 410, 50, 180, 221, 15, 348, 191, 225]

# Sample budgets for each scenario (replace with your actual budgets)
budgets = [100, 120, 80, 90]

# Sample probabilities for each scenario (replace with your actual probabilities)
probabilities = [0.55, 0.4, 0.04, 0.01]

# Create the Pyomo model
model = pyo.ConcreteModel()

# Define sets
model.strategies = pyo.RangeSet(1, m)
model.scenarios = pyo.RangeSet(1, scenarios)

# Define decision variables
model.x = pyo.Var(model.strategies, domain=pyo.Binary)
model.y = pyo.Var(model.scenarios, domain=pyo.Binary)

# Define parameters
model.returns = pyo.Param(model.strategies, initialize=lambda model, i: returns[i-1].mean())
model.costs = pyo.Param(model.strategies, initialize=lambda model, i: costs[i-1])
model.budgets = pyo.Param(model.scenarios, initialize=lambda model, k: budgets[k-1])
model.probabilities = pyo.Param(model.scenarios, initialize=lambda model, k: probabilities[k-1])

# Calculate covariance matrix
covariance_matrix = np.cov(returns)

# Define objective function
def objective_rule(model):
    return sum(model.returns[i] * model.x[i] for i in model.strategies) - \
           2 * sum(covariance_matrix[i-1, j-1] * model.x[i] * model.x[j] for i in model.strategies for j in model.strategies)

model.objective = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

# Define budget constraints for each scenario
def budget_constraint_rule(model, k):
    return sum(model.costs[i] * model.x[i] for i in model.strategies) <= model.budgets[k] + 1e6 * model.y[k]

model.budget_constraint = pyo.Constraint(model.scenarios, rule=budget_constraint_rule)

# Define probability of exceeding the budget constraint
def probability_constraint_rule(model):
    return sum(model.probabilities[k] * model.y[k] for k in model.scenarios) <= 0.4

model.probability_constraint = pyo.Constraint(rule=probability_constraint_rule)

# Define investment constraint
def investment_constraint_rule(model):
    return sum(model.x[i] for i in model.strategies) <= 1

model.investment_constraint = pyo.Constraint(rule=investment_constraint_rule)

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

# Print the results
print("Objective value:", pyo.value(model.objective))
print("Selected strategies:")
```

```
for i in model.strategies:
    if pyo.value(model.x[i]) > 0.5:
        print("Strategy", i)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-26-ed26b6c0b8b9> in <cell line: 69>()
     67 # Solve the model
     68 solver = pyo.SolverFactory('glpk')
---> 69 results = solver.solve(model)
     70 
     71 # Print the results

9 frames
/usr/local/lib/python3.10/dist-packages/pyomo/repn/plugins/lp_writer.py in
write(self, model)
    376         repn = objective_visitor.walk_expression(obj.expr)
    377         if repn.nonlinear is not None:
--> 378             raise ValueError(
    379                 f"Model objective ({obj.name}) contains nonlinear
terms that "
    380                 "cannot be written to LP format"

ValueError: Model objective (objective) contains nonlinear terms that cannot
```

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

```
%%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'


import pyomo.environ as pyo
import numpy as np
import pandas as pd

# Sample data (replace with your actual data)
scenarios = 4  # Number of budget scenarios

# Generate sample returns (replace with your actual returns)
data = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/IP/IP2/investments_data.csv", header=None, sep=';')
returns = data.to_numpy()
n, m = returns.shape

# Generate sample costs for each strategy (replace with your actual costs)
costs = [80, 340, 410, 50, 180, 221, 15, 348, 191, 225]

# Sample budgets for each scenario (replace with your actual budgets)
budgets = [100, 120, 80, 90]

# Sample probabilities for each scenario (replace with your actual probabilities)
probabilities = [0.55, 0.4, 0.04, 0.01]

# Create the Pyomo model
model = pyo.ConcreteModel()

# Define sets
model.strategies = pyo.RangeSet(1, m)
model.scenarios = pyo.RangeSet(1, scenarios)

# Define decision variables
model.x = pyo.Var(model.strategies, domain=pyo.Binary)
model.y = pyo.Var(model.scenarios, domain=pyo.Binary)

# Define parameters
model.returns = pyo.Param(model.strategies, initialize=lambda model, i: returns[i-1].mean())
model.costs = pyo.Param(model.strategies, initialize=lambda model, i: costs[i-1])
model.budgets = pyo.Param(model.scenarios, initialize=lambda model, k: budgets[k-1])
model.probabilities = pyo.Param(model.scenarios, initialize=lambda model, k: probabilities[k-1])

# Calculate covariance matrix
covariance_matrix = np.cov(returns)

# Define objective function
def objective_rule(model):
    return sum(model.returns[i] * model.x[i] for i in model.strategies) - \
           2 * sum(covariance_matrix[i-1, j-1] * model.x[i] * model.x[j] for i in model.strategies for j in model.strategies)
```

```
model.objective = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

# Define budget constraints for each scenario
def budget_constraint_rule(model, k):
    return sum(model.costs[i] * model.x[i] for i in model.strategies) <= model.budgets[k] + 1e6 * model.y[k]

model.budget_constraint = pyo.Constraint(model.scenarios, rule=budget_constraint_rule)

# Define probability of exceeding the budget constraint
def probability_constraint_rule(model):
    return sum(model.probabilities[k] * model.y[k] for k in model.scenarios) <= 0.4

model.probability_constraint = pyo.Constraint(rule=probability_constraint_rule)

# Define investment constraint
def investment_constraint_rule(model):
    return sum(model.x[i] for i in model.strategies) <= 1

model.investment_constraint = pyo.Constraint(rule=investment_constraint_rule)

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

# Print the results
print("Objective value:", pyo.value(model.objective))
print("Selected strategies:")
for i in model.strategies:
    if pyo.value(model.x[i]) > 0.5:
        print("Strategy", i)
```

```
Objective value: 0.0
Selected strategies:
```