0. Imports and Setting up Anthropic API Client

```
from google.colab import drive

drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

```
!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
```

+ Code + Text

```
!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 the mathematical optimization problem:

Decision variables:
x_j - binary variable indicating whether to invest in strategy j (x_j = 1) or not (x_j = 0), for j = 1, ..., 10

Parameters:
r_ij - historic return of strategy j in period i, for i = 1, ..., m (number of past periods), j = 1, ..., 10
A_j - cost of investing in strategy j, for j = 1, ..., 10
B_k - available budget in scenario k, for k = 1, ..., 4
p_k - probability of budget scenario k occurring, for k = 1, ..., 4
r - risk parameter, set to 2
α - tolerable probability of exceeding the budget, set to 0.4

Auxiliary variables:
y_k - binary variable indicating whether the budget is exceeded (y_k = 1) or not (y_k = 0) in scenario k, for k = 1, ...
μ_j - mean historic return of strategy j, for j = 1, ..., 10
σ_j - standard deviation of historic returns for strategy j, for j = 1, ..., 10
```

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)
```

```
Based on the provided information, here is the mean-variance mathematical model for the investment optimization problem,

Objective function:
Maximize ∑(j=1 to 10) μ_j * x_j - r * ∑(j=1 to 10) σ_j * x_j

Subject to:

Budget constraint for each scenario k:
∑(j=1 to 10) A_j * x_j - B_k * y_k ≤ B_k, for k = 1, ..., 4
```

```
Probability constraint for exceeding the budget:
∑(k=1 to 4) p_k * y_k ≤ α

Binary variable constraints:
x_j ∈ {0, 1}, for j = 1, ..., 10
y_k ∈ {0, 1}, for k = 1, ..., 4

where:
μ_j = (1/m) * ∑(i=1 to m) r_ij, for j = 1, ..., 10
σ_j = sqrt((1/m) * ∑(i=1 to m) (r_ij - μ_j)^2), for j = 1, ..., 10

The objective function aims to maximize the expected return of the selected investment strategies while minimizing the r

The budget constraints ensure that the total cost of the selected investment strategies does not exceed the available bu

The probability constraint limits the probability of exceeding the budget across all scenarios to the tolerable level α.

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

The mean historic return μ_j and the standard deviation of historic returns σ_j are calculated based on the past perform
```

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)
```

```
Thank you for providing the detailed problem description and the variables. Here are the constraints for the mean-varian

1. Budget constraint for each scenario k:
∑(j=1 to 10) A_j * x_j ≤ B_k + M * y_k, for k = 1, ..., 4

where M is a large positive number (Big M method) to ensure that if y_k = 1, the budget constraint is relaxed for scenar

2. Probability constraint for exceeding the budget:
∑(k=1 to 4) p_k * y_k ≤ α

3. Binary variable constraints:
x_j ∈ {0, 1}, for j = 1, ..., 10
y_k ∈ {0, 1}, for k = 1, ..., 4

4. Investment constraint (invest only once in each strategy):
x_j ≤ 1, for j = 1, ..., 10

Note: The mean historic return μ_j and the standard deviation of historic returns σ_j should be pre-calculated based on

μ_j = (1/m) * ∑(i=1 to m) r_ij, for j = 1, ..., 10
σ_j = sqrt((1/m) * ∑(i=1 to m) (r_ij - μ_j)^2), for j = 1, ..., 10

where m is the number of past periods (rows) in the data file.

These constraints, along with the objective function you mentioned, form the complete mean-variance mathematical optimiz
```

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 = """
Based on the provided information, here is the mean-variance mathematical model for the investment optimization problem, cor

Objective function:
Maximize ∑(j=1 to 10) μ_j * x_j - r * ∑(j=1 to 10) σ_j * x_j

where:
μ_j = (1/m) * ∑(i=1 to m) r_ij, for j = 1, ..., 10
σ_j = sqrt((1/m) * ∑(i=1 to m) (r_ij - μ_j)^2), for j = 1, ..., 10

The objective function aims to maximize the expected return of the selected investment strategies while minimizing the risk,

The mean historic return μ_j and the standard deviation of historic returns σ_j are calculated based on the past performance
"""



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 the Python Pyomo code for the investment optimization problem with the given mean-variance mathematical model

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

# Read investment data from CSV file
data = pd.read_csv("investments_data.csv", header=None)
data = data.to_numpy()

# Define problem parameters
m, n = data.shape
strategies = range(n)
scenarios = range(4)

# Sample data (replace with actual data)
A = np.random.randint(10, 50, size=n)  # Investment costs
B = np.random.randint(100, 500, size=4)  # Available budgets
p = np.random.dirichlet(np.ones(4))  # Scenario probabilities
r = 2  # Risk parameter
alpha = 0.4  # Tolerable probability of exceeding the budget

# Calculate mean and standard deviation of historic returns
mu = np.mean(data, axis=0)
sigma = np.std(data, axis=0)

# Create Pyomo model
model = pyo.ConcreteModel()

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

# Define objective function
model.obj = pyo.Objective(expr=sum(mu[j] * model.x[j] for j in strategies) -
                               r * sum(sigma[j] * model.x[j] for j in strategies),
                          sense=pyo.maximize)

# Define constraints
model.budget_con = pyo.ConstraintList()
for k in scenarios:
    model.budget_con.add(sum(A[j] * model.x[j] for j in strategies) <= B[k] + 1e6 * model.y[k])

model.prob_con = pyo.Constraint(expr=sum(p[k] * model.y[k] for k in scenarios) <= alpha)

model.investment_con = pyo.ConstraintList()
for j in strategies:
    model.investment_con.add(model.x[j] <= 1)

# Solve the optimization problem
solver = pyo.SolverFactory('cbc')
results = solver.solve(model)
```
```

```
    # Print the results
    print("Optimal investment strategy:")
```

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'

!pip install pandas
!pip install numpy
```

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

# Read investment data from CSV file
data = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/IP/IP2/investments_data.csv", header=None, sep=';')
data = data.to_numpy()

# Define problem parameters
m, n = data.shape
strategies = range(n)
scenarios = range(4)

# Sample data (replace with actual data)
A = [80, 340, 410, 50, 180, 221, 15, 348, 191, 225]
B = [1000, 1100, 900, 1200]  # Available budgets
p = [0.55, 0.4, 0.04, 0.01]  # Scenario probabilities
r = 2  # Risk parameter
alpha = 0.4  # Tolerable probability of exceeding the budget

# Calculate mean and standard deviation of historic returns
mu = np.mean(data, axis=0)
sigma = np.std(data, axis=0)

# Create Pyomo model
model = pyo.ConcreteModel()

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

# Define objective function
model.obj = pyo.Objective(expr=sum(mu[j] * model.x[j] for j in strategies) -
                               r * sum(sigma[j] * model.x[j] for j in strategies),
                          sense=pyo.maximize)

# Define constraints
model.budget_con = pyo.ConstraintList()
for k in scenarios:
    model.budget_con.add(sum(A[j] * model.x[j] for j in strategies) <= B[k] + 1e6 * model.y[k])

model.prob_con = pyo.Constraint(expr=sum(p[k] * model.y[k] for k in scenarios) <= alpha)

model.investment_con = pyo.ConstraintList()
for j in strategies:
    model.investment_con.add(model.x[j] <= 1)

# Solve the optimization problem
solver = pyo.SolverFactory('cbc')
results = solver.solve(model)

# Print the results
print("Optimal investment strategy:")
for j in strategies:
    if pyo.value(model.x[j]) == 1:
        print(f"Invest in strategy {j+1}")
for i in scenarios:
  if pyo.value(model.y[i]) == 1:
    print(f"Exceeded budget in scenario {i+1}")

print(f"\nOptimal objective value: {pyo.value(model.obj):.4f}")
```

```
print(f"\nOptimal objective value: {pyo.value(model.obj):.4f}")
```

```
Optimal investment strategy:
Invest in strategy 3
Invest in strategy 6
Invest in strategy 8
Exceeded budget in scenario 3

Optimal objective value: 237.8334
```

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