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)
```

```
Sure, I can help you formulate the variables for this mean-variance mathematical optimization problem. Here are the vari

Decision Variables:
- Let x_i be a binary decision variable, where x_i = 1 if we invest in strategy i, and x_i = 0 otherwise, for i = 1, 2,

Input Parameters:
- Let r_ij be the historic return of investment strategy i in scenario j, for i = 1, 2, ..., 10 and j = 1, 2, ..., n, wh
- Let c_i be the cost of investing in strategy i, for i = 1, 2, ..., 10. The values are given in vector A.
- Let B_k be the available budget in scenario k, for k = 1, 2, 3, 4. The values are 1000, 1100, 900, 1200 respectively.
- Let p_k be the probability of budget scenario k occurring, for k = 1, 2, 3, 4. The values are 0.55, 0.4, 0.04, 0.01 re
- Let α be the tolerable probability of exceeding the budget. The value is given as 0.4.
- Let r be the risk parameter. The value is given as 2.

Those are the key variables and parameters needed to formulate the mean-variance optimization model for selecting the op
```

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)
```

```
Sure, I can formulate the complete mean-variance optimization model for this investment strategy selection problem under

Objective Function:
Maximize  ∑_{i=1}^{10} μ_i x_i - r * ∑_{i=1}^{10} ∑_{j=1}^{10} σ_ij x_i x_j

where:
- μ_i is the mean historic return of investment strategy i, calculated as:
μ_i = (1/n) ∑_{j=1}^n r_ij
- σ_ij is the covariance between the returns of strategies i and j, calculated as:
σ_ij = (1/(n-1)) ∑_{k=1}^n (r_ik - μ_i)(r_jk - μ_j)

Constraints:
1. Budget constraints for each scenario k:
∑_{i=1}^{10} c_i x_i ≤ B_k + M y_k, for k = 1, 2, 3, 4
where M is a large positive number, and y_k are auxiliary binary variables
```

```
2. Probability constraint for exceeding budget:
∑_{k=1}^4 p_k y_k ≤ α

3. Invest only once in each strategy:
x_i ∈ {0, 1}, for i = 1, 2, ..., 10

4. Auxiliary variables:
y_k ∈ {0, 1}, for k = 1, 2, 3, 4

The objective function maximizes the mean return of the selected strategies minus a risk term multiplied by the paramete

Constraint 1 ensures the total cost of selected strategies does not exceed the budget in each scenario. The auxiliary va

Constraint 2 limits the probability of exceeding the budget to be no more than α, using the scenario probabilities p_k.

Constraint 3 enforces the binary nature of the decision variables x_i, allowing each strategy to be selected at most onc

Constraint 4 defines the auxiliary variables y_k as binary.

This mean-variance model balances maximizing expected return against minimizing risk, while accounting for the uncertain
```

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)
```

```
The mean-variance optimization model formulation looks good overall. Here are a few comments and suggestions:

1. In the objective function, it would be good to divide by n when calculating the covariance terms σ_ij, similar to how

   σ_ij = (1/n) ∑_{k=1}^n (r_ik - μ_i)(r_jk - μ_j)

2. For the budget constraints, you can simplify them slightly by moving the y_k terms to the right-hand side:

   ∑_{i=1}^{10} c_i x_i - M y_k ≤ B_k, for k = 1, 2, 3, 4

   This avoids adding the large M value to the budget value.

3. The other constraints look good. Just a small note - it's good practice to define the ranges of indices in the constr

4. One additional thing to consider: do you need a constraint to ensure at least one strategy is selected? If not invest

   ∑_{i=1}^{10} x_i ≥ 1

Overall, the model captures the key aspects of the problem - selecting strategies to maximize mean return minus a risk t
```

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_p11 + response_p12 + response_p13)}
    ]
)

response_p2 = message2.content[0].text


# Print response
print(response_p2)
```

Here's the Python code using Pyomo to formulate and solve the mean-variance optimization problem for selecting inves

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

# Sample data (replace with actual data)
n = 100  # Number of past scenarios
m = 10   # Number of investment strategies
r_ij = np.random.rand(m, n)  # Historic returns of strategies
c_i = np.random.rand(m)      # Costs of investing in strategies
B_k = [1000, 1100, 900, 1200]  # Available budgets in each scenario
p_k = [0.55, 0.4, 0.04, 0.01]  # Probabilities of budget scenarios
alpha = 0.4  # Tolerable probability of exceeding budget
r = 2        # Risk parameter

# Calculate mean returns and covariances
mu_i = np.mean(r_ij, axis=1)
cov_ij = np.cov(r_ij)

# Create Pyomo model
model = pyo.ConcreteModel()

# Define decision variables
model.x = pyo.Var(range(m), domain=pyo.Binary)
model.y = pyo.Var(range(4), domain=pyo.Binary)

# Define objective function
model.obj = pyo.Objective(expr=sum(mu_i[i] * model.x[i] for i in range(m)) -
                               r * sum(cov_ij[i, j] * model.x[i] * model.x[j] for i in range(m) for j in range(m)),
                          sense=pyo.maximize)

# Define constraints
model.budget_constr = pyo.ConstraintList()
for k in range(4):
    model.budget_constr.add(sum(c_i[i] * model.x[i] for i in range(m)) - 1e6 * model.y[k] <= B_k[k])

model.prob_constr = pyo.Constraint(expr=sum(p_k[k] * model.y[k] for k in range(4)) <= alpha)

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

# Print the results
print(f"Objective value: {pyo.value(model.obj):.4f}")
print("Selected strategies:")
for i in range(m):
    if pyo.value(model.x[i]) > 0.5:
        print(f"Strategy {i+1}")
```

Notes:
- I used numpy to generate random sample data for the historic returns (`r_ij`) and costs (`c_i`). Replace this with
- I calculated the mean returns (`mu_i`) and covariance matrix (`cov_ij`) using numpy functions.
- I defined the Pyomo model and variables, including the binary decision variables `x` and `y`.
- The objective function maximizes the mean return minus the risk term, using the covariance matrix.

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

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

# Sample data (replace with actual data)
data = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/IP/IP2/investments_data.csv", header=None, sep=';')
r_ij = data.to_numpy() # Historic returns of strategies
n, m = data.shape
c_i = [80, 340, 410, 50, 180, 221, 15, 348, 191, 225]   # Costs of investing in strategies
B_k = [1000, 1100, 900, 1200]  # Available budgets in each scenario
p_k = [0.55, 0.4, 0.04, 0.01]  # Probabilities of budget scenarios
alpha = 0.4  # Tolerable probability of exceeding budget
r = 2        # Risk parameter

# Calculate mean returns and covariances
```

```
mu_i = np.mean(r_ij, axis=1)
cov_ij = np.cov(r_ij)

# Create Pyomo model
model = pyo.ConcreteModel()

# Define decision variables
model.x = pyo.Var(range(m), domain=pyo.Binary)
model.y = pyo.Var(range(4), domain=pyo.Binary)

# Define objective function
model.obj = pyo.Objective(expr=sum(mu_i[i] * model.x[i] for i in range(m)) -
                               r * sum(cov_ij[i, j] * model.x[i] * model.x[j] for i in range(m) for j in range(m)),
                          sense=pyo.maximize)

# Define constraints
model.budget_constr = pyo.ConstraintList()
for k in range(4):
    model.budget_constr.add(sum(c_i[i] * model.x[i] for i in range(m)) - 1e6 * model.y[k] <= B_k[k])

model.prob_constr = pyo.Constraint(expr=sum(p_k[k] * model.y[k] for k in range(4)) <= alpha)

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

# Print the results
print(f"Objective value: {pyo.value(model.obj):.4f}")
print("Selected strategies:")
for i in range(m):
    if pyo.value(model.x[i]) > 0.5:
        print(f"Strategy {i+1}")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-1c18e5c8ec84> in <cell line: 40>()
     38 # Solve the model
     39 solver = pyo.SolverFactory('glpk')
---> 40 results = solver.solve(model)
     41 
     42 # 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 (obj) contains nonlinear terms that cannot be written to LP format
```

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 actual data)
data = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/IP/IP2/investments_data.csv", header=None, sep=';')
r_ij = data.to_numpy() # Historic returns of strategies
n, m = data.shape
c_i = [80, 340, 410, 50, 180, 221, 15, 348, 191, 225]   # Costs of investing in strategies
B_k = [1000, 1100, 900, 1200]  # Available budgets in each scenario
p_k = [0.55, 0.4, 0.04, 0.01]  # Probabilities of budget scenarios
alpha = 0.4  # Tolerable probability of exceeding budget
r = 2        # Risk parameter

# Calculate mean returns and covariances
mu_i = np.mean(r_ij, axis=1)
```

```
cov_ij = np.cov(r_ij)

# Create Pyomo model
model = pyo.ConcreteModel()

# Define decision variables
model.x = pyo.Var(range(m), domain=pyo.Binary)
model.y = pyo.Var(range(4), domain=pyo.Binary)

# Define objective function
model.obj = pyo.Objective(expr=sum(mu_i[i] * model.x[i] for i in range(m)) -
                               r * sum(cov_ij[i, j] * model.x[i] * model.x[j] for i in range(m) for j in range(m)),
                          sense=pyo.maximize)

# Define constraints
model.budget_constr = pyo.ConstraintList()
for k in range(4):
    model.budget_constr.add(sum(c_i[i] * model.x[i] for i in range(m)) - 1e6 * model.y[k] <= B_k[k])

model.prob_constr = pyo.Constraint(expr=sum(p_k[k] * model.y[k] for k in range(4)) <= alpha)

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

# Print the results
print(f"Objective value: {pyo.value(model.obj):.4f}")
print("Selected strategies:")
for i in range(m):
    if pyo.value(model.x[i]) > 0.5:
        print(f"Strategy {i+1}")
```

```
Objective value: 0.0000
Selected strategies:
```