## 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/IP/IP2.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:
 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: 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)
```

```
To formulate the mean-variance optimization model for this problem, we need to define the parameters, decision variables

Parameters:
- n: number of investment strategies (n = 10)
- m: number of past investment outcomes for each strategy
- R_ij: past investment outcome for strategy i in scenario j (i = 1, ..., n; j = 1, ..., m)
- A_i: cost of investing in strategy i (i = 1, ..., n)
- B_k: available budget in scenario k (k = 1, ..., 4)
- P_k: probability of budget scenario k (k = 1, ..., 4)
- α: tolerable probability of exceeding the budget (α = 0.4)
- r: risk parameter (r = 2)

Decision Variables:
- x_i: binary variable indicating whether to invest in strategy i (x_i = 1) or not (x_i = 0) (i = 1, ..., n)

Objective Function:
Maximize the mean return while minimizing the variance:
max ∑(i=1 to n) (μ_i * x_i) - r * ∑(i=1 to n) ∑(j=1 to n) (σ_ij * x_i * x_j)

where:
- μ_i = (1/m) * ∑(j=1 to m) R_ij (mean return for strategy i)
- σ_ij = (1/m) * ∑(j=1 to m) (R_ij - μ_i) * (R_ij - μ_j) (covariance between strategies i and j)

Constraints:
1. Budget constraint:
   ∑(i=1 to n) (A_i * x_i) ≤ B_k, for each budget scenario k (k = 1, ..., 4)

2. Probability constraint:
   ∑(k=1 to 4) (P_k * y_k) ≤ α, where y_k is a binary variable indicating whether the budget is exceeded in scenario k (

3. Linking constraint:
   ∑(i=1 to n) (A_i * x_i) - B_k ≤ M * y_k, for each budget scenario k (k = 1, ..., 4), where M is a large positive cons

4. Investment constraint:
   x_i ∈ {0, 1}, for each investment strategy i (i = 1, ..., n)

5. Probability indicator constraint:
   y_k ∈ {0, 1}, for each budget scenario k (k = 1, ..., 4)

This mean-variance optimization model aims to maximize the expected return while minimizing the variance, subject to the
```

## 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)
```

```
    model.y = pyo.Var(range(len(B)), domain=pyo.Binary)

    # Define the objective function
    def objective_rule(model):
        mean_return = sum(pyo.quicksum(R[i][j] for j in range(m)) / m * model.x[i] for i in range(n))
        variance = sum(
            sum(
                (1 / m) * ((R[i][j] - pyo.quicksum(R[i][j] for j in range(m)) / m) *
                           (R[k][j] - pyo.quicksum(R[k][j] for j in range(m)) / m)) *
                model.x[i] * model.x[k]
                for j in range(m)
            )
            for i in range(n) for k in range(n)
        )
        return mean_return - r * variance

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

    # Define the constraints
    def budget_constraint_rule(model, k):
        return sum(A[i] * model.x[i] for i in range(n)) <= B[k]

    model.budget_constraint = pyo.Constraint(range(len(B)), rule=budget_constraint_rule)

    def probability_constraint_rule(model):
        return sum(P[k] * model.y[k] for k in range(len(B))) <= alpha

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

    def linking_constraint_rule(model, k):
        return sum(A[i] * model.x[i] for i in range(n)) - B[k] <= M * model.y[k]

    model.linking_constraint = pyo.Constraint(range(len(B)), rule=linking_constraint_rule)

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

    # Print the results
    print("Objective value:", pyo.value(model.objective))
    print("Investment decisions:")
    for i in range(n):
        print("Strategy", i+1, ":", pyo.value(model.x[i]))
    ```

    In this code, sample data is used for the past investment outcomes (R), cost of investing in each strategy (A), availa
```

## 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 pandas as pd

# Define the model
model = pyo.ConcreteModel()

# Define the parameters
n = 10  # Number of investment strategies
m = 200  # Number of past investment outcomes for each strategy
r = 2  # Risk parameter
alpha = 0.4  # Tolerable probability of exceeding the budget

# Sample data (replace with actual data)
past_investments = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/IP/IP2/investments_data.csv", header=None, sep=';'
R = pd.DataFrame(past_investments).to_numpy()  # Past investment outcomes
R = R.T
A = [80, 340, 410, 50, 180, 221, 15, 348, 191, 225]  # Cost of investing in each strategy
B = [1000, 1100, 900, 1200]  # Available budget in each scenario
P = [0.55, 0.4, 0.04, 0.01]  # Probability of each budget scenario
M = sum(B)  # Large positive constant

# Define the decision variables
model.x = pyo.Var(range(n), domain=pyo.Binary)
model.y = pyo.Var(range(len(B)), domain=pyo.Binary)

# Define the objective function
def objective_rule(model):
    mean_return = sum(pyo.quicksum(R[i][j] for j in range(m)) / m * model.x[i] for i in range(n))
    variance = sum(
        sum(
            (1 / m) * ((R[i][j] - pyo.quicksum(R[i][j] for j in range(m)) / m) *
                       (R[k][j] - pyo.quicksum(R[k][j] for j in range(m)) / m)) *
            model.x[i] * model.x[k]
            for j in range(m)
        )
        for i in range(n) for k in range(n)
    )
    return mean_return - r * variance

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

# Define the constraints
def budget_constraint_rule(model, k):
    return sum(A[i] * model.x[i] for i in range(n)) <= B[k]

model.budget_constraint = pyo.Constraint(range(len(B)), rule=budget_constraint_rule)

def probability_constraint_rule(model):
    return sum(P[k] * model.y[k] for k in range(len(B))) <= alpha

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

def linking_constraint_rule(model, k):
    return sum(A[i] * model.x[i] for i in range(n)) - B[k] <= M * model.y[k]

model.linking_constraint = pyo.Constraint(range(len(B)), rule=linking_constraint_rule)

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

# Print the results
print("Objective value:", pyo.value(model.objective))
print("Investment decisions:")
for i in range(n):
    print("Strategy", i+1, ":", pyo.value(model.x[i]))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-6f6d2186f274> in <cell line: 60>()
     58 # Solve the model
     59 solver = pyo.SolverFactory('glpk')
---> 60 results = solver.solve(model)
     61 
     62 # 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)

```
# Installing pyomo nd solver
import sys
import os

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

Show hidden output

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

# Define the model
model = pyo.ConcreteModel()

# Define the parameters
n = 10  # Number of investment strategies
m = 200  # Number of past investment outcomes for each strategy
r = 2  # Risk parameter
alpha = 0.4  # Tolerable probability of exceeding the budget

# Sample data (replace with actual data)
past_investments = pd.read_csv("/content/drive/MyDrive/Thesis/ProblemData/IP/IP2/investments_data.csv", header=None, sep=';')
R = pd.DataFrame(past_investments).to_numpy()  # Past investment outcomes
R = R.T
A = [80, 340, 410, 50, 180, 221, 15, 348, 191, 225]  # Cost of investing in each strategy
B = [1000, 1100, 900, 1200]  # Available budget in each scenario
P = [0.55, 0.4, 0.04, 0.01]  # Probability of each budget scenario
M = sum(B)  # Large positive constant

# Define the decision variables
model.x = pyo.Var(range(n), domain=pyo.Binary)
model.y = pyo.Var(range(len(B)), domain=pyo.Binary)

# Define the objective function
def objective_rule(model):
    mean_return = sum(pyo.quicksum(R[i][j] for j in range(m)) / m * model.x[i] for i in range(n))
    variance = sum(
        sum(
            (1 / m) * ((R[i][j] - pyo.quicksum(R[i][j] for j in range(m)) / m) *
                       (R[k][j] - pyo.quicksum(R[k][j] for j in range(m)) / m)) *
            model.x[i] * model.x[k]
            for j in range(m)
        )
        for i in range(n) for k in range(n)
    )
    return mean_return - r * variance

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

# Define the constraints
def budget_constraint_rule(model, k):
    return sum(A[i] * model.x[i] for i in range(n)) <= B[k]

model.budget_constraint = pyo.Constraint(range(len(B)), rule=budget_constraint_rule)
```

```
def probability_constraint_rule(model):
    return sum(P[k] * model.y[k] for k in range(len(B))) <= alpha

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

def linking_constraint_rule(model, k):
    return sum(A[i] * model.x[i] for i in range(n)) - B[k] <= M * model.y[k]

model.linking_constraint = pyo.Constraint(range(len(B)), rule=linking_constraint_rule)

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

# Print the results
print("Objective value:", pyo.value(model.objective))
print("Investment decisions:")
for i in range(n):
```