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 period j (i = 1, ..., n; j = 1, ..., m)
- A_i: cost for investing in strategy i (i = 1, ..., n)
- r: risk parameter (r = 2)
- B_k: available budget in scenario k (k = 1, ..., 4)
- p_k: probability of scenario k (k = 1, ..., 4)
- α: tolerable probability of exceeding the budget (α = 0.4)

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:
The objective is to maximize the expected return while minimizing the risk, which can be formulated as:
max ∑(i=1 to n) (1/m × ∑(j=1 to m) R_ij) × x_i - r × sqrt(∑(i=1 to n) (1/m × ∑(j=1 to m) (R_ij - (1/m × ∑(j=1 to m) R_i

Constraints:
1. Budget constraint:
   P(∑(i=1 to n) A_i × x_i ≤ B_k) ≥ 1 - α, for k = 1, ..., 4
   This constraint ensures that the probability of the total investment cost not exceeding the available budget is at le

2. Investment constraint:
   x_i ∈ {0, 1}, for i = 1, ..., n
   This constraint ensures that each investment strategy is either selected (x_i = 1) or not (x_i = 0).

The complete mean-variance optimization model for this problem is:

max ∑(i=1 to n) (1/m × ∑(j=1 to m) R_ij) × x_i - r × sqrt(∑(i=1 to n) (1/m × ∑(j=1 to m) (R_ij - (1/m × ∑(j=1 to m) R_i

subject to:
1. P(∑(i=1 to n) A_i × x_i ≤ B_k) ≥ 1 - α, for k = 1, ..., 4
2. x_i ∈ {0, 1}, for i = 1, ..., n

This model aims to find the optimal investment strategy selection that maximizes the expected return while considering 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_p1}
    ]
)

response_p2 = message2.content[0].text


# Print response
print(response_p2)
```

```
# Parameters
n = 10  # number of investment strategies
m = 5   # 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)
R = np.random.rand(n, m)  # past investment outcomes
```

```
    # Decision variables
    model.x = pyo.Var(range(n), within=pyo.Binary)

    # Objective function
    model.obj = pyo.Objective(expr=sum((1/m * sum(R[i,j] for j in range(m))) * model.x[i] for i in range(n))
                              - r * pyo.sqrt(sum((1/m * sum((R[i,j] - (1/m * sum(R[i,j] for j in range(m))))**2 for j in r
                              sense=pyo.maximize)

    # Constraints
    def budget_constraint(model, k):
        return pyo.summation(A, model.x) <= B[k]

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

    def probability_constraint(model):
        return sum(p[k] * model.budget[k].body for k in range(len(B))) >= 1 - alpha

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

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

    # Print the results
    print("Objective value:", pyo.value(model.obj))
    print("Selected strategies:")
    for i in range(n):
        if pyo.value(model.x[i]) == 1:
            print("Strategy", i+1)
    ```

    In this code:
    - The parameters are defined based on the problem description. Sample data is used for the past investment outcomes (`
    - The Pyomo model is created using `pyo.ConcreteModel()`.
    - The decision variables `x` are defined as binary variables using `pyo.Var()` with the `within=pyo.Binary` option.
    - The objective function is defined using `pyo.Objective()` with the expression from the problem formulation.
    - The budget constraint is defined using `pyo.Constraint()` and a rule function `budget_constraint()`. It ensures that
    - The probability constraint is defined using `pyo.Constraint()` and a rule function `probability_constraint()`. It er
    - The model is solved using the GLPK solver (`glpk`).
    - The objective value and selected strategies are printed.

    Note: Make sure you have Pyomo and GLPK installed to run this code. You can install them using `pip install pyomo` anc
```

## 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

# 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()
R = R.T
A = [80, 340, 410, 50, 180, 221, 15, 348, 191, 225]  # cost for 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 scenario

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

# Decision variables
model.x = pyo.Var(range(n), within=pyo.Binary)

# Objective function
model.obj = pyo.Objective(expr=sum((1/m * sum(R[i,j] for j in range(m))) * model.x[i] for i in range(n))
                          - r * pyo.sqrt(sum((1/m * sum((R[i,j] - (1/m * sum(R[i,j] for j in range(m))))**2 for j in range(m)
                          sense=pyo.maximize)
```
```

```
# Constraints
def budget_constraint(model, k):
    return pyo.summation(A, model.x) <= B[k]

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

def probability_constraint(model):
    return sum(p[k] * model.budget[k].body for k in range(len(B))) >= 1 - alpha

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

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

# Print the results
print("Objective value:", pyo.value(model.obj))
print("Selected strategies:")
for i in range(n):
    if pyo.value(model.x[i]) == 1:
        print("Strategy", i+1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-20-88ff2b71fd16> in <cell line: 42>()
     40 # Solve the model
     41 solver = pyo.SolverFactory('glpk')
---> 42 results = solver.solve(model)
     43 
     44 # 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
```

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

# 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()
R = R.T
A = [80, 340, 410, 50, 180, 221, 15, 348, 191, 225]  # cost for 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 scenario

# Small value to prevent sqrt from evaluating zero
epsilon = 1e-6 # MODIFIED TO INCLUDE EPSILON

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

# Decision variables
model.x = pyo.Var(range(n), within=pyo.Binary)
```

```
    # Objective function
    model.obj = pyo.Objective(expr=sum((1/m * sum(R[i,j] for j in range(m))) * model.x[i] for i in range(n))
                              - r * pyo.sqrt(sum((1/m * sum((R[i,j] - (1/m * sum(R[i,j] for j in range(m))))**2 for j in range(m
                              sense=pyo.maximize)

    # Constraints
    def budget_constraint(model, k):
        return pyo.summation(A, model.x) <= B[k]

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

    def probability_constraint(model):
        return sum(p[k] * model.budget[k].body for k in range(len(B))) >= 1 - alpha

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

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

    # Print the results
    print("Objective value:", pyo.value(model.obj))
    print("Selected strategies:")
    for i in range(n):
        if pyo.value(model.x[i]) == 1:
            print("Strategy", i+1)
```

```
Objective value: 217.08352160678973
Selected strategies:
Strategy 1
Strategy 6
Strategy 8
Strategy 10
```