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
```

```
Collecting anthropic
  Downloading anthropic-0.26.0-py3-none-any.whl (877 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 877.7/877.7 kB 6.6 MB/s eta 0:00:00
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-packages (from anthropic) (1.7.0)
Collecting httpx<1,>=0.23.0 (from anthropic)
  Downloading httpx-0.27.0-py3-none-any.whl (75 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 10.1 MB/s eta 0:00:00
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (2.7.1)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from anthropic) (1.3.1)
Requirement already satisfied: tokenizers>=0.13.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (0.19.1)
Requirement already satisfied: typing-extensions<5,>=4.7 in /usr/local/lib/python3.10/dist-packages (from anthropic) (4.
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->anthropic) (3
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->anthropi
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->anthropic) (20
Collecting httpcore==1.* (from httpx<1,>=0.23.0->anthropic)
  Downloading httpcore-1.0.5-py3-none-any.whl (77 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 8.8 MB/s eta 0:00:00
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->anthropic)
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 6.7 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.
Requirement already satisfied: pydantic-core==2.18.2 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.0
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/lib/python3.10/dist-packages (from tokenizers>
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->t
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->t
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->huggi
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface
Installing collected packages: h11, httpcore, httpx, anthropic
Successfully installed anthropic-0.26.0 h11-0.14.0 httpcore-1.0.5 httpx-0.27.0
```

```
# 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: historic return of strategy i in outcome j (i = 1, ..., N; j = 1, ..., M)
- A_i: cost of investing in strategy i (i = 1, ..., N)
- S: number of budget scenarios (S = 4)
- B_s: available budget in scenario s (s = 1, ..., S)
- p_s: probability of scenario s (s = 1, ..., S)
- 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)
- y_s: binary variable indicating whether the budget is exceeded in scenario s (y_s = 1) or not (y_s = 0) (s = 1, ..., S

Objective Function:
Maximize the mean return minus the variance of the portfolio:
max ∑(i=1 to N) (1/M × ∑(j=1 to M) R_ij) × x_i - r × (∑(i=1 to N) (1/M × ∑(j=1 to M) (R_ij - (1/M × ∑(j=1 to M) R_ij))^2

Constraints:
1. Budget constraint for each scenario s:
   ∑(i=1 to N) A_i × x_i ≤ B_s + M × y_s, for s = 1, ..., S
   where M is a large positive number (big M method)

2. Probability of exceeding the budget:
   ∑(s=1 to S) p_s × y_s ≤ α

3. Binary constraints:
   x_i ∈ {0, 1}, for i = 1, ..., N
   y_s ∈ {0, 1}, for s = 1, ..., S

In this model, the objective function maximizes the mean return of the portfolio minus the variance multiplied by the ri

Note: The historic returns R_ij should be read from the "investments_data.csv" file, with each column representing a str
```

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)
```

```
# Define the parameters
N = 10
M = R.shape[0]
A = np.array([1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500])  # Sample cost data
S = 4
B = np.array([10000, 15000, 20000, 25000])  # Sample budget data
p = np.array([0.2, 0.3, 0.3, 0.2])  # Sample probability data
alpha = 0.4
r = 2

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

# Define the decision variables
model.x = pyo.Var(range(N), domain=pyo.Binary)
model.y = pyo.Var(range(S), domain=pyo.Binary)

# Define the objective function
def obj_rule(model):
    mean_return = sum((1/M * sum(R[:, i])) * model.x[i] for i in range(N))
    variance = sum((1/M * sum((R[:, i] - (1/M * sum(R[:, i]))) ** 2)) * model.x[i] for i in range(N))
    return mean_return - r * variance

model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)

# Define the constraints
def budget_constraint_rule(model, s):
    return sum(A[i] * model.x[i] for i in range(N)) <= B[s] + 1e6 * model.y[s]

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

def probability_constraint_rule(model):
    return sum(p[s] * model.y[s] for s in range(S)) <= alpha

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

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

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

In this code:
- We assume that the historic returns data is stored in a CSV file named "investments_data.csv", with each column repr
- We define the parameters N, M, A, S, B, p, alpha, and r. The values for A, B, and p are sample data, which should be
- We create a Pyomo ConcreteModel and define the decision variables x and y.
- We define the objective function, which maximizes the mean return minus the variance multiplied by the risk paramete
- We define the budget constraint for each scenario s using the big M method, where 1e6 is used as a large positive nu
- We define the probability constraint, which limits the probability of exceeding the budget to alpha.
- We solve the model using the GLPK solver (you may need to install it separately).
- Finally, we print the objective value and the selected investments.

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

# Read the historic returns data from the CSV file
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

# Define the parameters
N = 10
M = R.shape[0]
A =[80, 340, 410, 50, 180, 221, 15, 348, 191, 225]  # Sample cost data
S = 4
B = [1000, 1100, 900, 1200]   # Sample budget data
p = [0.55, 0.4, 0.04, 0.01]  # Sample probability data
alpha = 0.4
r = 2

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

# Define the decision variables
model.x = pyo.Var(range(N), domain=pyo.Binary)
model.y = pyo.Var(range(S), domain=pyo.Binary)

# Define the objective function
def obj_rule(model):
    mean_return = sum((1/M * sum(R[:, i])) * model.x[i] for i in range(N))
    variance = sum((1/M * sum((R[:, i] - (1/M * sum(R[:, i])))) ** 2)) * model.x[i] for i in range(N))
    return mean_return - r * variance

model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)

# Define the constraints
def budget_constraint_rule(model, s):
    return sum(A[i] * model.x[i] for i in range(N)) <= B[s] + 1e6 * model.y[s]

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

def probability_constraint_rule(model):
    return sum(p[s] * model.y[s] for s in range(S)) <= alpha

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

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

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

```
Objective value: 226.08424825247408
Selected investments:
Investment 3
Investment 6
Investment 8
```

```
for i in range(S):
    if pyo.value(model.y[i]) == 1:
        print("Scenario", i+1)
```

```
Scenario 3
```

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