*import* yfinance *as* yf

*import* pandas *as* pd

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*import* cvxpy *as* cp

*# Define the tickers and download the data*

tickers = ["SPY", "GOLD", "NDX"]

data = yf.download(tickers, *start*="2020-01-01", *end*="2024-01-01", *auto\_adjust*=True)

*# Calculate the daily returns*

returns = data['Close'].pct\_change().dropna()

*# Calculate the expected returns and the sample covariance*

expected\_returns = returns.mean()

sample\_cov = returns.cov()

*# Number of assets*

n = len(tickers)

*# Define the portfolio weights as variables*

weights = cp.Variable(n)

*# Define the portfolio return*

portfolio\_return = expected\_returns.values @ weights

*# Define the portfolio risk (standard deviation)*

portfolio\_risk = cp.quad\_form(weights, sample\_cov.values)

*# Define the constraints*

constraints = [cp.sum(weights) == 1, weights >= 0]

*# Define the objective function (minimize risk for a given level of return)*

risk\_free\_rate = 0.01 / 252 *# Assuming a daily risk-free rate*

target\_return = cp.Parameter()

objective = cp.Minimize(portfolio\_risk)

*# Define the problem*

problem = cp.Problem(objective, constraints + [portfolio\_return >= target\_return])

*# Solve the problem for a range of target returns to find the optimal Sharpe ratio*

target\_returns = np.linspace(expected\_returns.min(), expected\_returns.max(), 50)

sharpe\_ratios = []

*for* target *in* target\_returns:

target\_return.value = target

problem.solve()

*if* problem.status == cp.OPTIMAL:

risk = cp.sqrt(portfolio\_risk).value

sharpe\_ratio = (target - risk\_free\_rate) / risk

sharpe\_ratios.append(sharpe\_ratio)

*else*:

sharpe\_ratios.append(None) *# Handle infeasible problems*

*# Find the optimal weights for the maximum Sharpe ratio*

optimal\_index = np.argmax([s *for* s *in* sharpe\_ratios *if* s is not None])

optimal\_return = target\_returns[optimal\_index]

target\_return.value = optimal\_return

problem.solve()

optimal\_weights = weights.value

optimal\_risk = cp.sqrt(portfolio\_risk).value

*# Print the optimal weights*

print("Optimal Weights:", optimal\_weights)

*# Calculate the expected return and risk of the optimal portfolio*

optimal\_return = np.dot(expected\_returns.values, optimal\_weights)

optimal\_risk = np.sqrt(np.dot(optimal\_weights.T, np.dot(sample\_cov.values, optimal\_weights)))

print("Expected Return of Optimal Portfolio:", optimal\_return)

print("Risk of Optimal Portfolio:", optimal\_risk)

*# Plot the efficient frontier*

def **efficient\_frontier**(*returns*, *cov\_matrix*):

n = len(*returns*)

target\_returns = np.linspace(*returns*.min(), *returns*.max(), 50)

efficient\_portfolios = []

*for* target *in* target\_returns:

weights = cp.Variable(n)

portfolio\_return = *returns*.values @ weights

portfolio\_risk = cp.quad\_form(weights, *cov\_matrix*.values)

constraints = [cp.sum(weights) == 1, weights >= 0, portfolio\_return == target]

objective = cp.Minimize(portfolio\_risk)

problem = cp.Problem(objective, constraints)

problem.solve()

*if* problem.status == cp.OPTIMAL:

efficient\_portfolios.append((target, np.sqrt(problem.value), weights.value))

*return* efficient\_portfolios

efficient\_portfolios = efficient\_frontier(expected\_returns, sample\_cov)

*# Plotting*

plt.figure(*figsize*=(10, 6))

plt.plot([p[0] *for* p *in* efficient\_portfolios], [p[1] *for* p *in* efficient\_portfolios], 'y-o', *label*='Efficient Frontier')

plt.scatter(optimal\_return, optimal\_risk, *color*='red', *label*='Optimal Portfolio')

plt.xlabel('Expected Return')

plt.ylabel('Risk (Standard Deviation)')

plt.title('Efficient Frontier and Optimal Portfolio')

plt.legend()

plt.grid(True)

plt.show()