NOMURA GLOBAL MARKETS' WOMEN MENTORSHIP PROGRAM

AYUSHI MEHTA

IIT KANPUR

HIGHLIGHTS OF THE MENTORSHIP PROGRAM

- Delved into the world of finance especially stock markets under the esteemed guidance of experienced professionals at Nomura.
- Researched and worked on two projects :

  1. Pricing European Options using Monte Carlo Simulations
  2. Equity Markets

PRICING EUROPEAN OPTIONS USING MONTE CARLO SIMULATIONS

MONTE CARLO SIMULATIONS

- It is a powerful computational technique used to forecast future stock prices by generating numerous possible price paths.
- It involves random sampling and statistical modeling to account for the uncertainty in stock price movements.
- By simulating a wide range of potential outcomes, analysts can better understand the possible future behaviors of stock prices, assess risks, and make more informed investment decisions.

GEOMETRIC BROWNIAN MOTION

- Assuming that stock price follow a lognormal distribution where the stock returns follow a normal distribution, we simulate the stock prices using the Geometric Brownian motion equation.

Stochastic Differential Equation (SDE):

$$dS_t = \mu S_t\,dt + \sigma S_t\,dW_t$$

- S_t: Stock price at time t
- μ: Drift (expected return)
- σ: Volatility (standard deviation of returns)
- dW_t: Wiener process (Brownian motion)

Solution:

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right)$$

- S_0: Initial stock price at $t = 0$
- $\left(\mu - \frac{\sigma^2}{2}\right)t$: Deterministic trend
- σW_t: Stochastic component (randomness)

```python
def simulate_stock_prices(S0, mu, sigma, T, dt, N, M):
    Z = np.random.standard_normal((N, M))  # standard normal variables
    S = np.zeros((N + 1, M))  # array to store simulated stock prices
    S[0] = S0  # initial stock price
    for t in range(1, N + 1):
        S[t] = S[t-1] * np.exp((mu - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * Z[t-1])
    return S
```

PAYOFF AT MATURITY

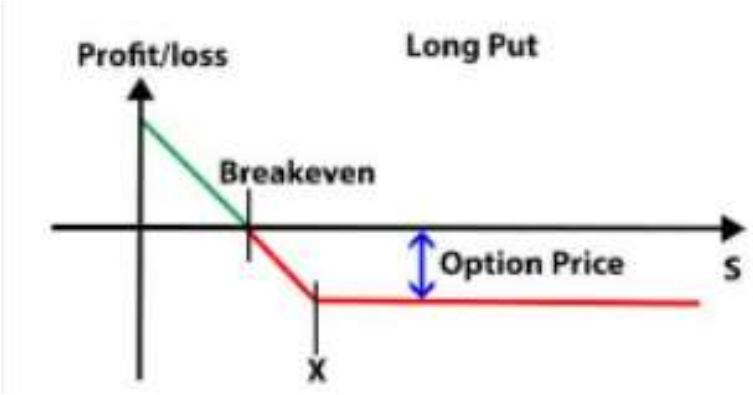
- **Definition:** The payoff at maturity for options refers to the amount of money an investor gains or loses when an options contract expires, based on the difference between the market price of the underlying asset and the option's strike price.
- There are two kinds of options ‘Call Option’ and ‘Put Option’ and the formula for calculating payoff at maturity varies slightly for both of them.

CALL OPTION

- **Definition:** A call option gives the holder the right, but not the obligation, to buy an underlying asset at a specified strike price on or before the option's expiration date.
- **Payoff Formula:** $\text{Payoff} = \max(S_T - K, 0)$
- **Formula for pricing a European Call Option:** $C = E(S_{\{T\}} - K)^+ * e^{-rT}$

```
def price_call_option(S, K, r, T):
    payoffs = np.maximum(S[-1] - K, 0)  # payoff at maturity
    return np.exp(-r * T) * np.mean(payoffs)  # discounted average payoff
```

PUT OPTION



- **Definition**: A put option gives the holder the right, but not the obligation, to sell an underlying asset at a specified strike price on or before the option's expiration date.
- **Payoff Formula:** $\text{Payoff} = \max(K - S_T, 0)$
- **Formula for pricing a European Put Option:** $C = E(K - S_{\{T\}})^{+} * e^{-rT}$

```
def price_put_option(S, K, r, T):
    payoffs = np.maximum(K - S[-1], 0)  # payoff at maturity
    return np.exp(-r * T) * np.mean(payoffs)  # discounted average payoff
```

SIGNIFICANCE OF FORMULA USED FOR PRICING A EUROPEAN OPTION

$$C = E(S_{\{T\}} - K)^{+} * e^{-rT}$$

1. $E[(S_T - K)^+]$: The term $E[(S_T - K)^+]$ represents the average expected payoff of the option at maturity, considering the randomness in stock price movements.
2. e^{-rT}: This factor accounts for the time value of money. Money today is worth more than the same amount in the future due to the potential earning capacity. By discounting the expected payoff at the risk-free rate, we determine what that future payoff is worth in today's terms.

BLACK-SCHOLES FORMULA FOR EUROPEAN OPTIONS

- The Black-Scholes formula provides a theoretical estimate of the price of European-style options, which can only be exercised at expiration.

Call Option Price (C):

$$C = S_0\mathcal{N}(d_1) - Ke^{-rT}\mathcal{N}(d_2)$$

Put Option Price (P):

$$P = Ke^{-rT}\mathcal{N}(d_2) - S_0\mathcal{N}(d_1)$$

Intermediate Calculations:

$$d_1 = \frac{\ln(S_0/K)+(r+\sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

$$N(d) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{d}\exp(-t^2/2)\,dt$$

Where:

- S_0 = Current stock price
- K = Strike price
- T = Time to maturity (in years)
- r = Risk-free interest rate (annual)
- σ = Volatility of the stock's returns (annual standard deviation)
- $\mathcal{N}(x)$ = Cumulative distribution function of the standard normal distribution

CODE SNIPPET FOR VERIFICATION USING BLACK-SCHOLES FORMULA

```
# Verification using Black-Scholes formula
def black_scholes_call(S0, K, T, r, sigma):
    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    call_price = (S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2))
    return call_price

def black_scholes_put(S0, K, T, r, sigma):
    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    put_price = (K * np.exp(-r * T) * norm.cdf(-d2) - S0 * norm.cdf(-d1))
    return put_price
```

RESULT

- Values taken - K : 90, T : 0.5

- The price of the European call option is: 14.57
- The price of the European put option is: 3.66
- Verified call price using Black-Scholes formula: 14.58
- Verified put price using Black-Scholes formula: 3.69

DELTA

Definition:

Delta measures the sensitivity of an option's price to changes in the price of the underlying asset. It represents the rate of change of the option's price with respect to a one-unit change in the underlying asset's price.

Formula:

$$\Delta = \frac{\partial C}{\partial S}$$

Where:

- C is the price of the option.
- S is the price of the underlying asset.

VEGA

Definition:

Vega measures the sensitivity of an option's price to changes in the volatility of the underlying asset. Specifically, it quantifies how much the price of an option will change with a 1% change in the volatility of the underlying asset.

Formula:

$$\text{Vega} = \frac{\partial C}{\partial \sigma}$$

Where:

- C is the price of the option.
- σ is the volatility of the underlying asset.

CODE SNIPPETS FOR DELTA AND VEGA

```
dS = 1.0  # small change in stock price
dSigma = 0.01*sigma  # small change in volatility
```

```
# Calculate delta for call
S_up = simulate_stock_prices(S0 + dS, mu, sigma, T, dt, N, M)
call_price_up = price_call_option(S_up, K, r, T)
delta_call = (call_price_up - call_price) / dS
print(f"Delta of the call option: {delta_call:.4f}")

# Calculate delta for put
put_price_up = price_put_option(S_up, K, r, T)
delta_put = (put_price_up - put_price) / dS
print(f"Delta of the put option: {delta_put:.4f}")
```

```
# Calculate vega for call
S_sigma_up = simulate_stock_prices(S0, mu, sigma + dSigma, T, dt, N, M)
call_price_sigma_up = price_call_option(S_sigma_up, K, r, T)
vega_call = (call_price_sigma_up - call_price) / dSigma
print(f"Vega of the call option: {vega_call:.4f}")

# Calculate vega for put
put_price_sigma_up = price_put_option(S_sigma_up, K, r, T)
vega_put = (put_price_sigma_up - put_price) / dSigma
print(f"Vega of the put option: {vega_put:.4f}")
```

CALCULATED VALUES OF DELTA AND VEGA

- Values taken - K : 90, T : 0.5

- Delta of the call option: 0.7469
- Delta of the put option: -0.2177
- Vega of the call option: 28.2172
- Vega of the put option: 36.5257

USING DIFFERENT STRIKES AND TENOR

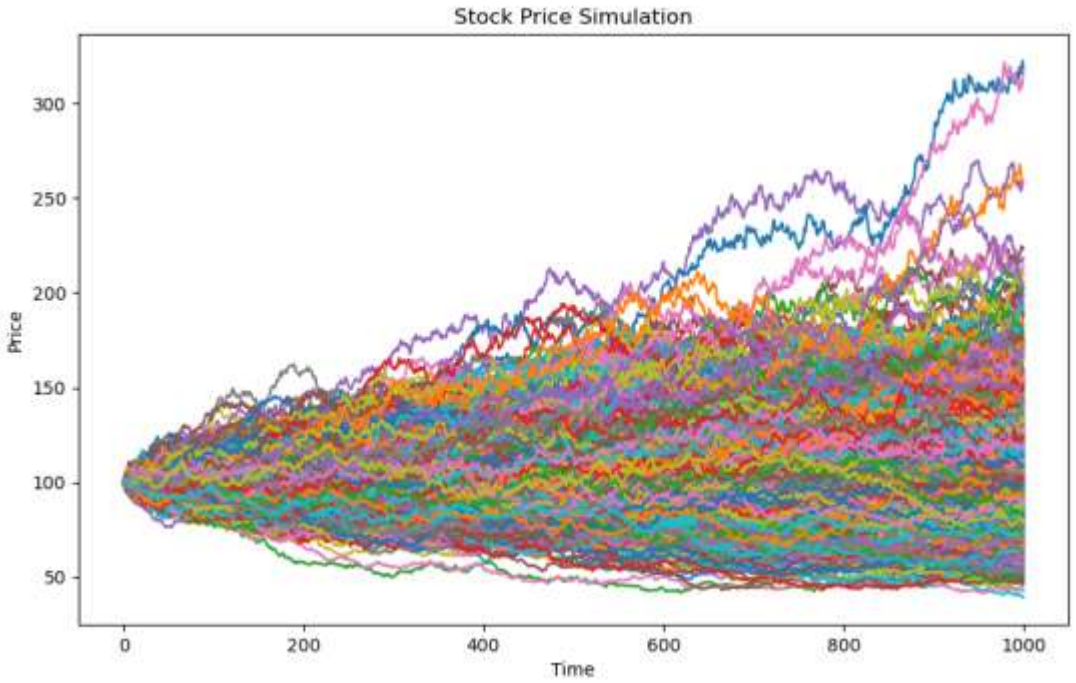
```
strike_prices = [90, 100, 110]
tenors = [0.5, 1, 2]
```

```
K : 90
T : 0.5
The price of the European call option is: 14.53
The price of the European put option is: 3.70
Verified call price using Black-Scholes formula: 14.58
Verified put price using Black-Scholes formula: 3.69
Delta of the call option: 0.6917
Delta of the put option: -0.2251
Vega of the call option: 47.1941
Vega of the put option: 12.3903
----------------------------
----------------------------
K : 90
T : 1
The price of the European call option is: 18.00
The price of the European put option is: 6.32
Verified call price using Black-Scholes formula: 18.07
Verified put price using Black-Scholes formula: 6.29
Delta of the call option: 0.9440
Delta of the put option: -0.3279
Vega of the call option: 74.6750
Vega of the put option: 28.9250
----------------------------
----------------------------
K : 90
T : 2
The price of the European call option is: 23.34
The price of the European put option is: 9.82
Verified call price using Black-Scholes formula: 23.32
Verified put price using Black-Scholes formula: 9.79
Delta of the call option: 0.8129
Delta of the put option: -0.3185
Vega of the call option: 17.9556
Vega of the put option: 36.7374
```

```
K : 100
T : 0.5
The price of the European call option is: 8.89
The price of the European put option is: 7.92
Verified call price using Black-Scholes formula: 8.91
Verified put price using Black-Scholes formula: 7.92
Delta of the call option: 0.6104
Delta of the put option: -0.3952
Vega of the call option: 22.3636
Vega of the put option: 38.7138
----------------------------
----------------------------
K : 100
T : 1
The price of the European call option is: 12.89
The price of the European put option is: 10.82
Verified call price using Black-Scholes formula: 12.82
Verified put price using Black-Scholes formula: 10.84
Delta of the call option: 0.5742
Delta of the put option: -0.3398
Vega of the call option: 105.3210
Vega of the put option: 29.0814
----------------------------
----------------------------
K : 100
T : 2
The price of the European call option is: 18.34
The price of the European put option is: 14.62
Verified call price using Black-Scholes formula: 18.50
Verified put price using Black-Scholes formula: 14.58
Delta of the call option: 0.7527
Delta of the put option: -0.5078
Vega of the call option: 107.8504
Vega of the put option: 42.0459
```

```
K : 110
T : 0.5
The price of the European call option is: 5.07
The price of the European put option is: 13.96
Verified call price using Black-Scholes formula: 5.07
Verified put price using Black-Scholes formula: 13.98
Delta of the call option: 0.4758
Delta of the put option: -0.6633
Vega of the call option: 13.8840
Vega of the put option: 58.6870
----------------------------
----------------------------
K : 110
T : 1
The price of the European call option is: 8.88
The price of the European put option is: 16.66
Verified call price using Black-Scholes formula: 8.86
Verified put price using Black-Scholes formula: 16.69
Delta of the call option: 0.4584
Delta of the put option: -0.4772
Vega of the call option: 69.3685
Vega of the put option: 55.0230
----------------------------
----------------------------
K : 110
T : 2
The price of the European call option is: 14.51
The price of the European put option is: 20.28
Verified call price using Black-Scholes formula: 14.58
Verified put price using Black-Scholes formula: 20.27
Delta of the call option: 0.5476
Delta of the put option: -0.4479
Vega of the call option: 138.7996
Vega of the put option: 9.7276
```

VISUALIZING SIMULATIONS




```
# Parameters
S0 = 100  # initial stock price
K = 100   # strike price
T = 1     # time to maturity (1 year)
r = 0.02  # risk-free ann rate
sigma = 0.30  # volatility
mu = 0.02  # expected return == ann rfr
dt = 0.001  # time increment
N = int(T / dt)  # number of steps
M = 100000  # number of simulations
```

Parameters Used

$$S_{\{t\}} = S_{\{t-dt\}} * e^{\left(\mu - \frac{\sigma^2}{2}\right) * dt + \sigma * dw}$$

$$dw = \sqrt{dt} * z$$

Stochastic GBM Equation

EQUITY MARKETS

DEFINITION

- Equity markets, also known as stock markets or share markets, are platforms where shares of publicly traded companies are bought and sold.
- These markets enable investors to purchase ownership stakes in companies, which are represented by shares of stock.
- The main functions of equity markets include providing companies with access to capital and giving investors the opportunity to participate in the financial performance of those companies.

NIFTY 50 INDEX

- The NIFTY 50 is a stock market index representing the weighted average of 50 of the largest and most liquid Indian companies listed on the National Stock Exchange of India (NSE).
- It serves as a benchmark index for the Indian equity market, providing a measure of the overall performance of the market and the economy.

TOP 10 LARGE CAP STOCKS IN NIFTY 50 INDEX TAKEN (AS ON 15.04.2024)

1. BAJFINANCE
2. HDFCBANK
3. ICICIBANK
4. INFY
5. ITC
6. KOTAKBANK
7. LT
8. RELIANCE
9. SBIN
10. TCS

BETA OF A STOCK

Definition: Beta (ß) is a measure of a stock's volatility or systematic risk in comparison to the overall market. It indicates how much the stock's price is expected to move relative to market movements. A beta value can help investors understand the risk associated with a particular stock in the context of the market.

Interpretation of Beta Values:

- **ß = 1**: The stock's price is expected to move in line with the market.
- **ß > 1**: The stock is more volatile than the market. For example, a beta of 1.5 means the stock is expected to be 50% more volatile than the market.
- **ß < 1**: The stock is less volatile than the market. For example, a beta of 0.5 means the stock is expected to be 50% less volatile than the market.
- **ß = 0**: The stock's price is not correlated with the market movements.
- **ß < 0**: The stock moves in the opposite direction to the market.

CALCULATING BETA

1. Using Formula: $\beta = \frac{\text{Cov}(R_i, R_m)}{\text{Var}(R_m)}$

   Where:

   - β: Beta of the stock
   - R_i: Return of the stock
   - R_m: Return of the market
   - $\text{Cov}(R_i, R_m)$: Covariance between the stock's returns and the market's returns
   - $\text{Var}(R_m)$: Variance of the market's returns

2. Using Linear Regression: Slope of the best-fit line in the plot of ‘Return of Stock VS Return of Market’

PYTHON CODE AND OUTPUT

1. Calculating Returns:

```
# Calculate daily returns
returns = data.pct_change().dropna()
nifty_returns = nifty_data.pct_change().dropna()

# Align returns
aligned_returns = returns.join(nifty_returns.rename('NIFTY'), how='inner').dropna()
X = sm.add_constant(aligned_returns['NIFTY'])
aligned_returns.pop('NIFTY')
```

2. Calculating Beta values of each stock:

```
# Calculate Beta values
betas = {}
for stock in aligned_returns.columns:
    y = aligned_returns[stock]
    model = sm.OLS(y, X).fit()
    betas[stock] = model.params[1]
```

Output:

```
Stock Betas: {'BAJFINANCE.NS': 1.4315941782234214, 'HDFCBANK.NS':
1.0975542368529714, 'ICICIBANK.NS': 1.3261752624057936, 'INFY.NS':
0.853470250290903, 'ITC.NS': 0.6892305332137675, 'KOTAKBANK.NS':
1.0528805454758812, 'LT.NS': 1.0408947050483996, 'RELIANCE.NS':
1.0840669903244806, 'SBIN.NS': 1.2570260335803172, 'TCS.NS':
0.7147265663689188}
```

SELECTION OF TIME FRAME

- The choice of time period for calculating Beta, expected returns using CAPM, and constructing the efficient frontier is pivotal.
- A common approach is to use the last 5 years of returns.
- This period is long enough to capture a variety of market conditions, including bull and bear markets, yet recent enough to be relevant to current conditions.

```
# Define the stocks
stocks = ['RELIANCE.NS', 'INFY.NS', 'HDFCBANK.NS', 'KOTAKBANK.NS', 'TCS.NS',
          'BAJFINANCE.NS', 'SBIN.NS', 'ITC.NS', 'LT.NS', 'ICICIBANK.NS']

# Define the time period for historical data
end_date = datetime.today()
start_date = end_date - timedelta(days=5*365)
```

WEIGHTED AVERAGE BETA OF A PORTFOLIO

1. **Determine the Market Capitalization Weights**: Calculate the weight of each stock in the portfolio based on its market capitalization relative to the total market capitalization of all stocks in the portfolio.
2. **Calculate the Weighted Beta**: Multiply each stock's beta by its corresponding weight.
3. **Sum the Weighted Betas**: Add up all the weighted betas to get the portfolio's weighted average beta.

Formula:

$$\beta_p = \sum_{i=1}^{n} w_i \beta_i$$

Where:

- β_p: Weighted average beta of the portfolio
- w_i: Weight of the i-th stock in the portfolio, calculated as $w_i = \frac{\text{Market Cap of Stock } i}{\text{Total Market Cap of Portfolio}}$
- β_i: Beta of the i-th stock
- n: Number of stocks in the portfolio

PYTHON CODE AND OUTPUT

1. Calculating Weights using Market Capitalization

```
# Fetch market capitalization for each stock
market_caps = {stock: yf.Ticker(stock).info['marketCap'] for stock in stocks}

# Calculate total market capitalization and weights
total_market_cap = sum(market_caps.values())
weights = np.array([market_caps[stock] / total_market_cap for stock in stocks])
```

2. Calculating Portfolio Beta

```
# Calculate portfolio Beta
portfolio_beta = sum(betas[stock] * weight for stock, weight in zip(betas, weights))
```

Output

```
Portfolio Beta: 1.0947337634097793
```

CAPITAL ASSET PRICING MODEL (CAPM)

Definition: The Capital Asset Pricing Model (CAPM) is a financial model used to determine the expected return of an asset based on its systematic risk (beta), the risk-free rate, and the expected market return. The CAPM formula is used to calculate the expected return of an asset or a portfolio of assets.

CAPM Formula:

$$E(R_i) = R_f + \beta_i(E(R_m) - R_f)$$

Where:

- $E(R_i)$: Expected return of the asset i
- R_f: Risk-free rate (e.g., the yield on government bonds)
- β_i: Beta of the asset i
- $E(R_m)$: Expected return of the market
- $E(R_m) - R_f$: Market risk premium (the additional return expected from holding a risky market portfolio instead of risk-free assets)

PYTHON CODE AND OUTPUT

```
# Risk-free rate and market risk premium
risk_free_rate = 0.03
market_risk_premium = 0.08
```

1. Calculating expected returns of each stock

```
# Calculate expected returns using CAPM
expected_returns = {stock: risk_free_rate + betas[stock] * market_risk_premium for stock in betas}
```

2. Calculating Portfolio expected return

```
# Calculate portfolio expected return
portfolio_expected_return = risk_free_rate + portfolio_beta * market_risk_premium
```

Outputs:

```
Expected Returns: {'BAJFINANCE.NS': 0.1445275342578737,
'HDFCBANK.NS': 0.1178043389482377, 'ICICIBANK.NS':
0.13609402099246348, 'INFY.NS': 0.09827762002327224, 'ITC.NS':
0.0851384426571014, 'KOTAKBANK.NS': 0.1142304436380705, 'LT.NS':
0.11327157640387198, 'RELIANCE.NS': 0.11672535922595845, 'SBIN.NS':
0.13056208268642538, 'TCS.NS': 0.08717812530951351}

Portfolio Expected Return: 0.11757870107278234
```

SHARPE RATIO

Definition: The Sharpe Ratio is a measure of risk-adjusted return. It indicates how much excess return an investment generates for each unit of risk taken. The higher the Sharpe Ratio, the better the investment's risk-adjusted performance.

Formula:

$$\text{Sharpe Ratio} = \frac{E(R_p) - R_f}{\sigma_p}$$

Where:

- $E(R_p)$: Expected return of the portfolio or investment
- R_f: Risk-free rate (e.g., the yield on government bonds)
- σ_p: Standard deviation of the portfolio's returns (a measure of total risk)

EXPLANATION OF FORMULA

$$\frac{E(R_p) - R_f}{\sigma_p}$$

- **Excess Return** $(E(R_p) - R_f)$: This represents the return of the portfolio above the risk-free rate. It's the additional return earned for taking on risk.
- **Standard Deviation** (σ_p) **:** This measures the volatility or risk of the portfolio's returns. It captures the degree to which the portfolio's returns can vary.
- **Risk-Adjusted Performance**: By dividing the excess return by the standard deviation, the Sharpe Ratio provides a way to compare investments on a risk-adjusted basis. Investments with higher Sharpe Ratios offer better risk-adjusted returns.

FINDING OPTIMAL WEIGHTS

```python
# Portfolio optimization for maximum Sharpe Ratio
cov_matrix = aligned_returns.cov()*252

def neg_sharpe_ratio(weights, cov_matrix, risk_free_rate, betas):
    p_var = np.dot(weights.T, np.dot(cov_matrix, weights))
    wbeta = sum(betas[stock] * weight for stock, weight in zip(betas, weights))
    p_ret = wbeta*0.08 + 0.03
    return -(p_ret - risk_free_rate) / np.sqrt(p_var)

constraints = ({'type': 'eq', 'fun': (lambda x: (np.sum(x) - 1))})
bounds = tuple((0, 1) for _ in range(len(stocks)))
initial_guess = 10 * [1. / 10]

opt_results = minimize(neg_sharpe_ratio, initial_guess, args=(cov_matrix, risk_free_rate, betas), method='SLSQP',
                       bounds=bounds, constraints=constraints)

optimal_weights = opt_results.x
print(opt_results)
```

Output :

```
Optimal Weights: [0.00000000e+00 9.86404876e-17 1.12643642e-01
1.63630185e-01
 5.78560234e-03 1.35639180e-16 3.36086721e-01 1.63306104e-01
 7.48588057e-02 1.43688941e-01]
```

CALCULATIONS FOR THE OPTIMAL PORTFOLIO

```
# Calculate the expected return, volatility, and Sharpe Ratio of the optimal portfolio

portfolio_beta_new = sum(betas[stock] * weight for stock, weight in zip(betas, optimal_weights))
opt_ret = risk_free_rate + portfolio_beta_new * market_risk_premium
opt_vol = np.sqrt(np.dot(optimal_weights.T, np.dot(cov_matrix, optimal_weights)))
opt_sharpe_ratio = (opt_ret - risk_free_rate) / opt_vol
```

Output :

```
Expected Annual Return: 0.11133517881266652
Annual Volatility/Risk: 0.2094968015147577
Sharpe Ratio: 0.38824067109653215
```

EFFICIENT FRONTIER

- The efficient frontier is a concept in modern portfolio theory introduced by Harry Markowitz.
- It represents a set of optimal portfolios that offer the highest expected return for a defined level of risk or the lowest risk for a given level of expected return.
- These portfolios are considered efficient because no other portfolio can provide a better risk-return combination.

GENERATING PORTFOLIOS FOR THE EFFICIENT FRONTIER

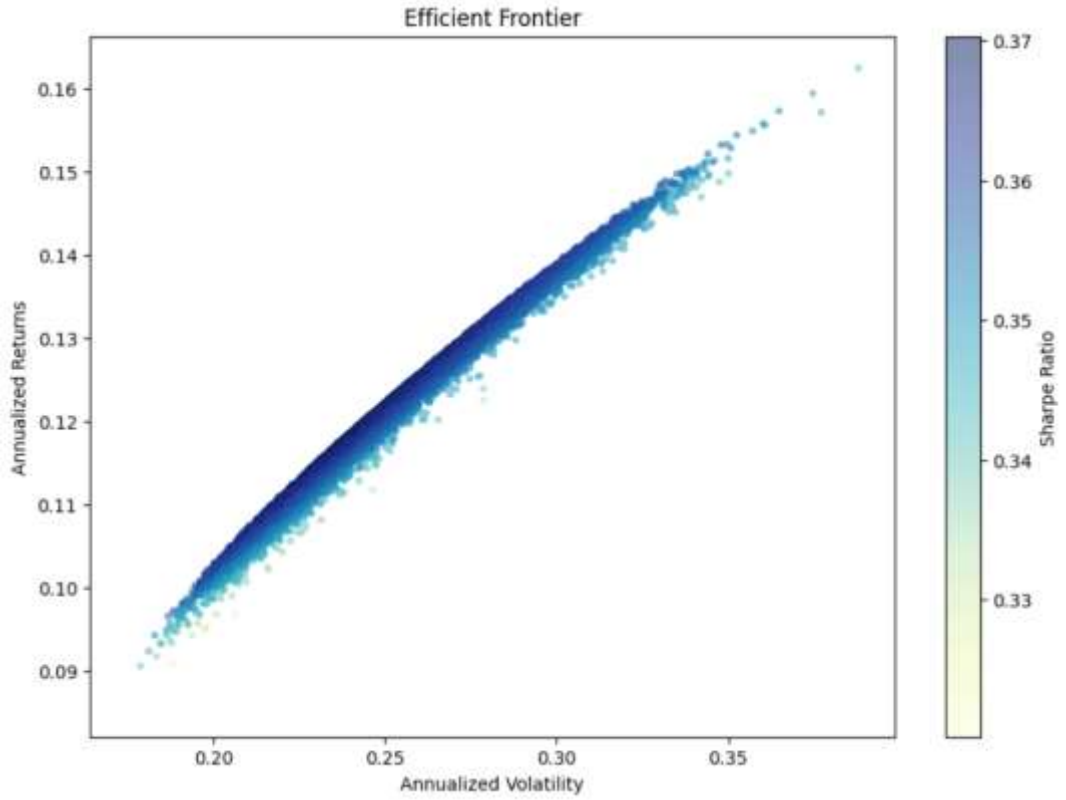
```python
# Generating portfolios for the efficient frontier
def generate_portfolios(num_portfolios=25000):
    results = np.zeros((3, num_portfolios))
    for i in range(num_portfolios):
        weights = np.random.random(len(stocks))
        weights /= np.sum(weights)
        portfolio_std_dev, portfolio_return, _ = portfolio_performance(weights, mean_returns, cov_matrix, risk_free_rate, betas)
        results[0, i] = portfolio_std_dev
        results[1, i] = portfolio_return
        results[2, i] = (portfolio_return - risk_free_rate) / portfolio_std_dev  # Sharpe Ratio
    return results

def portfolio_performance(weights, mean_returns, cov_matrix, risk_free_rate, betas):
    portfolio_beta = sum(betas[stock] * weight for stock, weight in zip(betas, weights))
    returns = risk_free_rate + portfolio_beta * market_risk_premium
    # returns = np.sum(mean_returns * weights)
    std_dev = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    sharpe_ratio = (returns - risk_free_rate) / std_dev
    return std_dev, returns, sharpe_ratio

results = generate_portfolios()
```

PLOTTING EFFICIENT FRONTIER

```
# Plotting the efficient frontier
plt.figure(figsize=(10, 7))
plt.scatter(results[0], results[1], c=results[2], cmap='YlGnBu', marker='o', s=10, alpha=0.5)
plt.colorbar(label='Sharpe Ratio')
plt.title('Efficient Frontier')
plt.xlabel('Annualized Volatility')
plt.ylabel('Annualized Returns')
plt.show()
```



THANK YOU