Assignment_2

Bidit Sadhukhan

Anirban Dey

Soumyadeep Sadhukhan

Import Libraries

```
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
```

Define a function to download historical price data

```
def download_historical_data(stocks, start_date, end_date):
    data = {}
    for sector, symbols in stocks.items():
        sector_data = pd.DataFrame()
        for symbol in symbols:
            stock = yf.download(symbol, start=start_date,
end=end_date)
            returns = stock['Adj Close'].pct_change().dropna()
            sector_data[symbol] = returns
        data[sector] = sector_data
    return data
```

Define a function to calculate mean and variance of daily returns

```
def calculate_statistics(data):
    mu_hat = {}
    v_hat = {}
    for sector, sector_data in data.items():
        mu_hat[sector] = sector_data.mean()
```

```
        v_hat[sector] = sector_data.var()
    return mu_hat, v_hat
```

Define a function for portfolio optimization

```
def portfolio_optimization(data, mu_hat, b):
    w_b = {}
    for sector, sector_data in data.items():
        cov_matrix = sector_data.cov()
        ones_vector = np.ones(len(stocks[sector]))
        inv_cov_matrix = np.linalg.inv(cov_matrix)

        w_b[sector] = np.dot(inv_cov_matrix, mu_hat[sector] - b *
ones_vector) / np.sum(np.dot(inv_cov_matrix, ones_vector))
    return w_b
```

Define a function for simulating portfolio performance and calculating MSE

```
def simulate_portfolio_performance(data, w_b, mu_hat,
months_to_simulate):
    results = {}
    for month in months_to_simulate:
        results[month] = []
        for sector, sector_data in data.items():
            try:
                port_returns = np.dot(sector_data.values, w_b[sector])
                overall_port_returns = np.dot(port_returns,
np.ones(len(port_returns)))
                mse = np.mean((overall_port_returns - mu_hat[sector])
** 2)
                results[month].append(mse)
            except ValueError as e:
                print(f"Error processing {sector}: {e}")
    return results
```

Define a function to compare results with initial return

```
def compare_initial_mse(data, w_b, mu_hat):
    initial_mse = {}
    for sector, sector_data in data.items():
```

```
        try:
            initial_returns = np.dot(sector_data.iloc[0], w_b[sector])
            initial_mse[sector] = (initial_returns - mu_hat[sector]
[0]) ** 2
        except ValueError as e:
            print(f"Error processing {sector}: {e}")
    return initial_mse
```

Main code

```
if __name__ == "__main__":
    # List of stock symbols
    stocks = {
        'Pharma': ['AAPL', 'MSFT'],
        'Banking': ['JPM', 'BAC'],
        'Technology': ['GOOGL', 'AMZN'],
        'Agriculture': ['CAG', 'ADM'],
        'Others': ['TSLA', 'NFLX']
    }

    # Define date range for historical data
    start_date = "2021-01-01"
    end_date = "2023-01-01"

    # Download historical price data
    data = download_historical_data(stocks, start_date, end_date)

    # Calculate mean and variance of daily returns
    mu_hat, v_hat = calculate_statistics(data)
    print("MU_HAT=\n",mu_hat)
    print("V_HAT=\n",v_hat)
    # Portfolio optimization
    b = 0.001
    w_b = portfolio_optimization(data, mu_hat, b)

    # Simulate portfolio performance and calculate mean squared error
    months_to_simulate = [1, 3, 6]
    results = simulate_portfolio_performance(data, w_b, mu_hat,
months_to_simulate)

    # Compare results with initial return
    initial_mse = compare_initial_mse(data, w_b, mu_hat)

    # Print or analyze the results as needed
    print("Results for 1 month:", results[1])
    print("Results for 3 months:", results[3])
    print("Results for 6 months:", results[6])
    print("Initial MSE:", initial_mse)
```

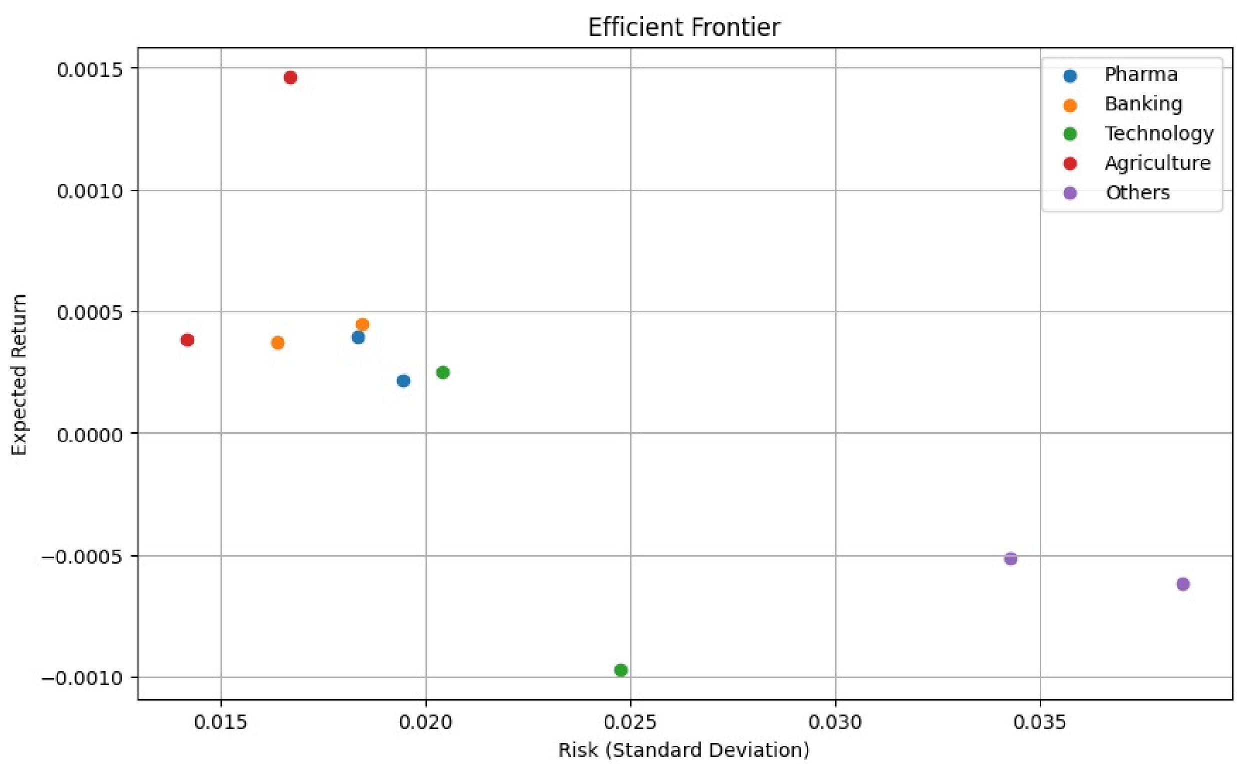
```
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
MU_HAT=
 {'Pharma': AAPL    0.000220
MSFT    0.000396
dtype: float64, 'Banking': JPM    0.000371
BAC    0.000448
dtype: float64, 'Technology': GOOGL    0.000252
AMZN    -0.000968
dtype: float64, 'Agriculture': CAG    0.000386
ADM    0.001464
dtype: float64, 'Others': TSLA   -0.000616
NFLX   -0.000512
dtype: float64}
V_HAT=
 {'Pharma': AAPL    0.000378
MSFT    0.000336
dtype: float64, 'Banking': JPM    0.000268
BAC    0.000340
dtype: float64, 'Technology': GOOGL    0.000417
AMZN     0.000613
dtype: float64, 'Agriculture': CAG    0.000201
ADM    0.000278
dtype: float64, 'Others': TSLA    0.001482
NFLX    0.001175
dtype: float64}
Results for 1 month: [1.5848900465833216e-07, 2.6593051666412515e-07,
1.946711517707269e-06, 8.716448123004669e-07, 1.0017137203894836e-06]
Results for 3 months: [1.5848900465833216e-07, 2.6593051666412515e-07,
1.946711517707269e-06, 8.716448123004669e-07, 1.0017137203894836e-06]
Results for 6 months: [1.5848900465833216e-07, 2.6593051666412515e-07,
1.946711517707269e-06, 8.716448123004669e-07, 1.0017137203894836e-06]
Initial MSE: {'Pharma': 5.168131571935987e-08, 'Banking':
1.3957445444911803e-07, 'Technology': 6.9108980178491e-08,
'Agriculture': 1.3882629099217177e-07, 'Others': 3.774858009832006e-
07}
```

Calculate portfolios on the efficient frontier

```
returns = np.array(list(mu_hat.values()))
risk = np.sqrt(list(v_hat.values()))

plt.figure(figsize=(10, 6))
for i, sector in enumerate(data.keys()):
    plt.scatter(risk[i], returns[i], label=sector, marker='o')

plt.title('Efficient Frontier')
plt.xlabel('Risk (Standard Deviation)')
plt.ylabel('Expected Return')
plt.legend()
plt.grid(True)
plt.show()
```



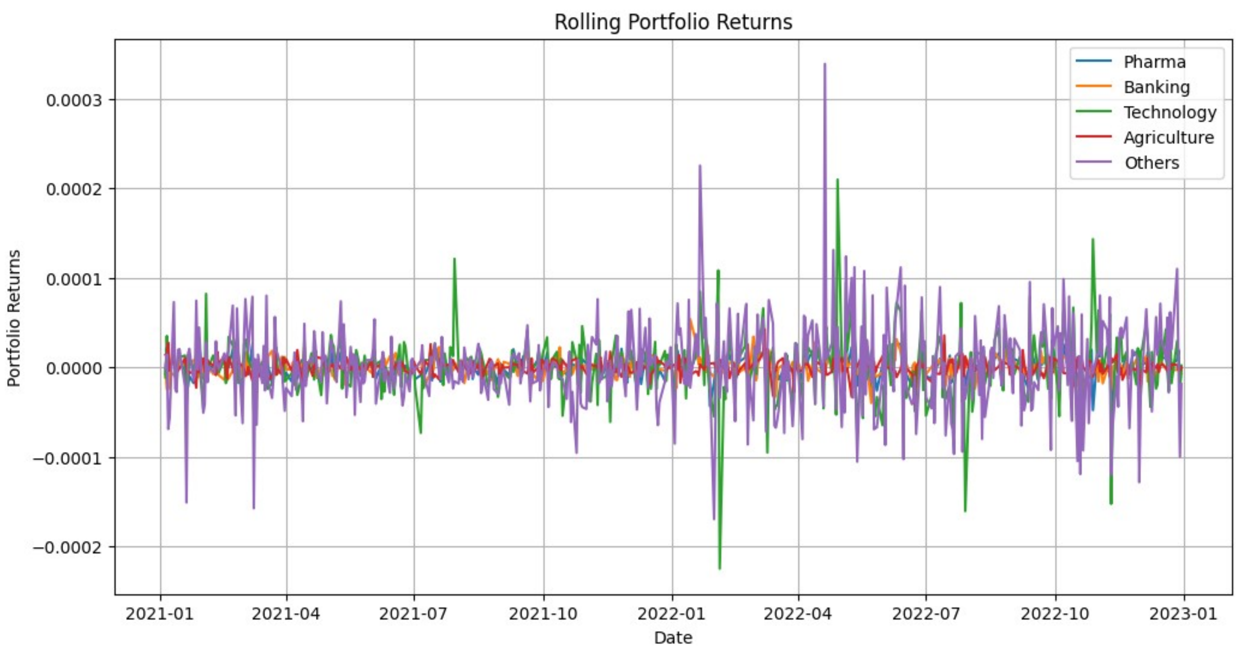
Calculate rolling portfolio returns

```
rolling_returns = pd.DataFrame()

for sector, sector_data in data.items():
    rolling_returns[sector] = sector_data.apply(lambda x: np.dot(x,
w_b[sector]), axis=1)
```

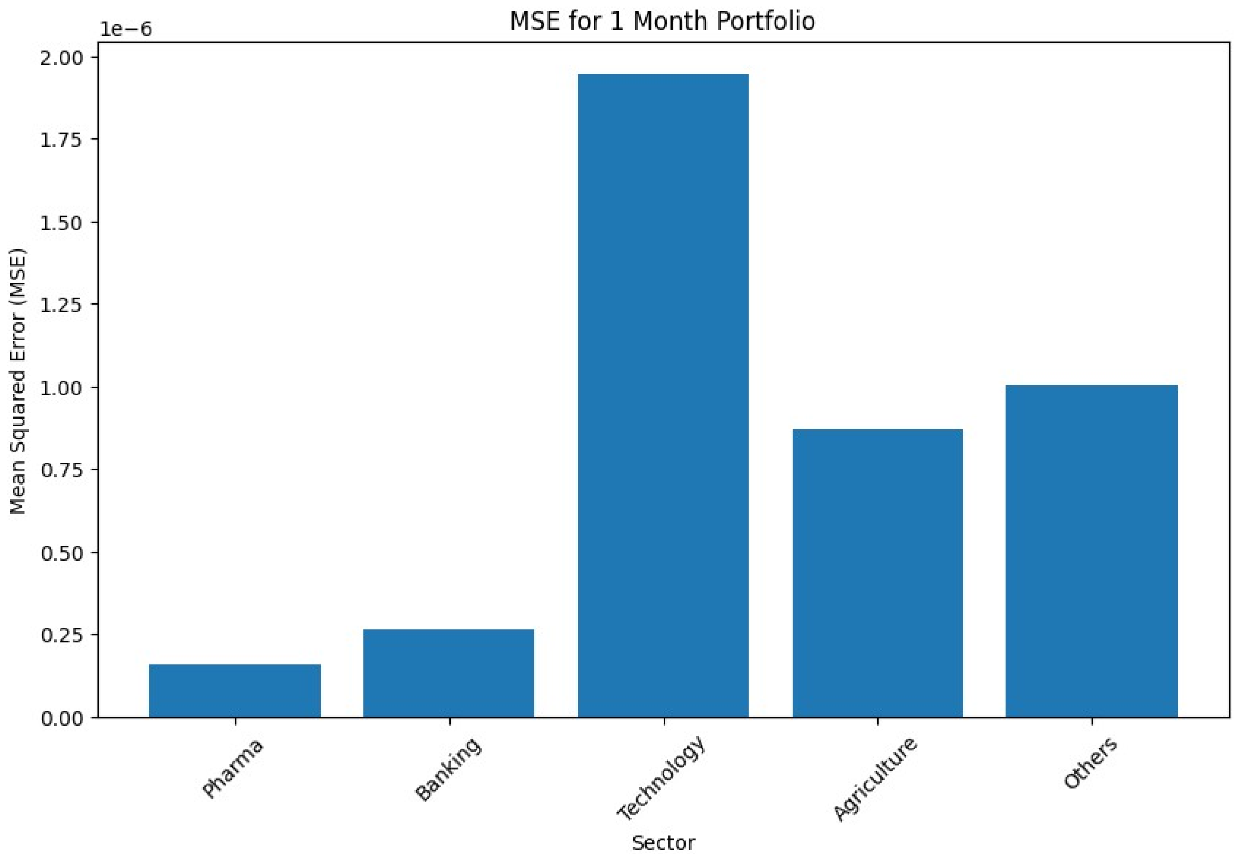
```
plt.figure(figsize=(12, 6))
for sector in data.keys():
    plt.plot(rolling_returns.index, rolling_returns[sector],
label=sector)

plt.title('Rolling Portfolio Returns')
plt.xlabel('Date')
plt.ylabel('Portfolio Returns')
plt.legend()
plt.grid(True)
plt.show()
```

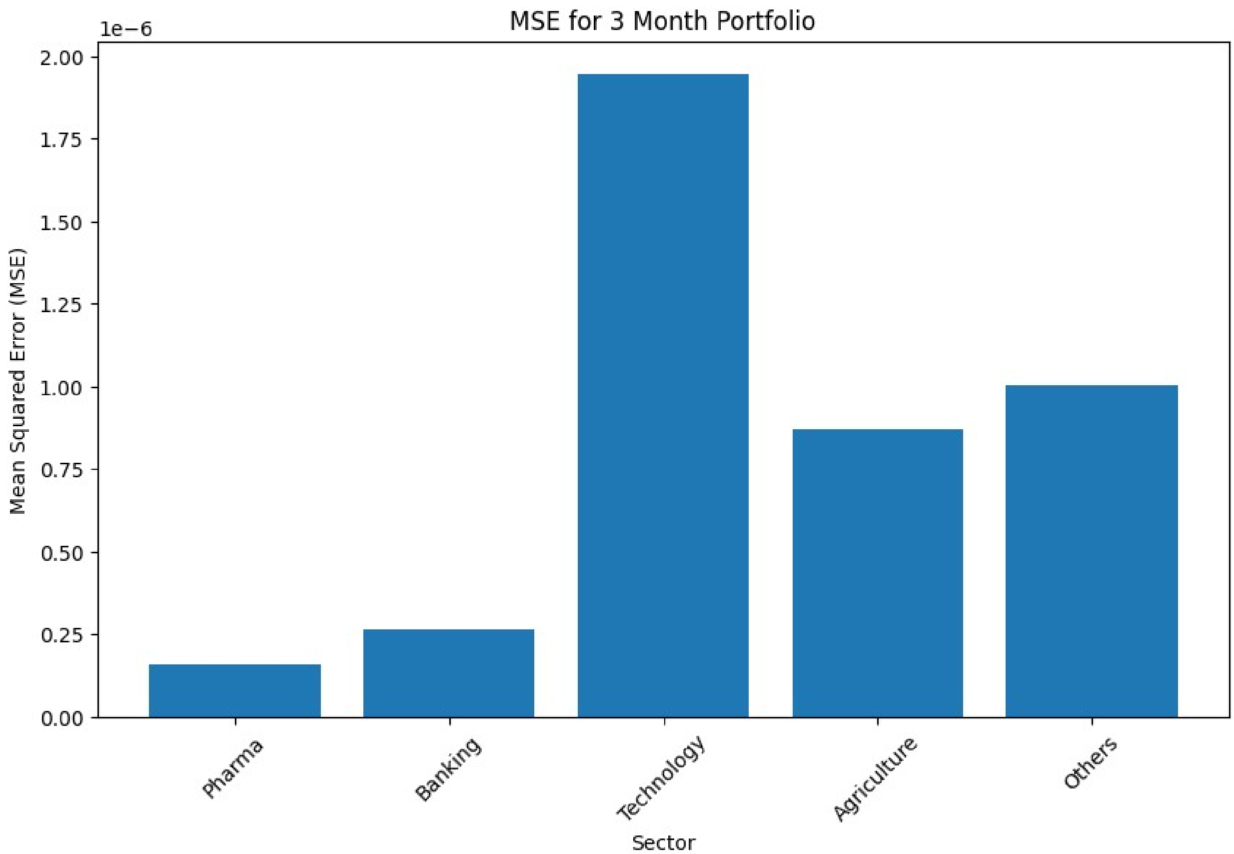


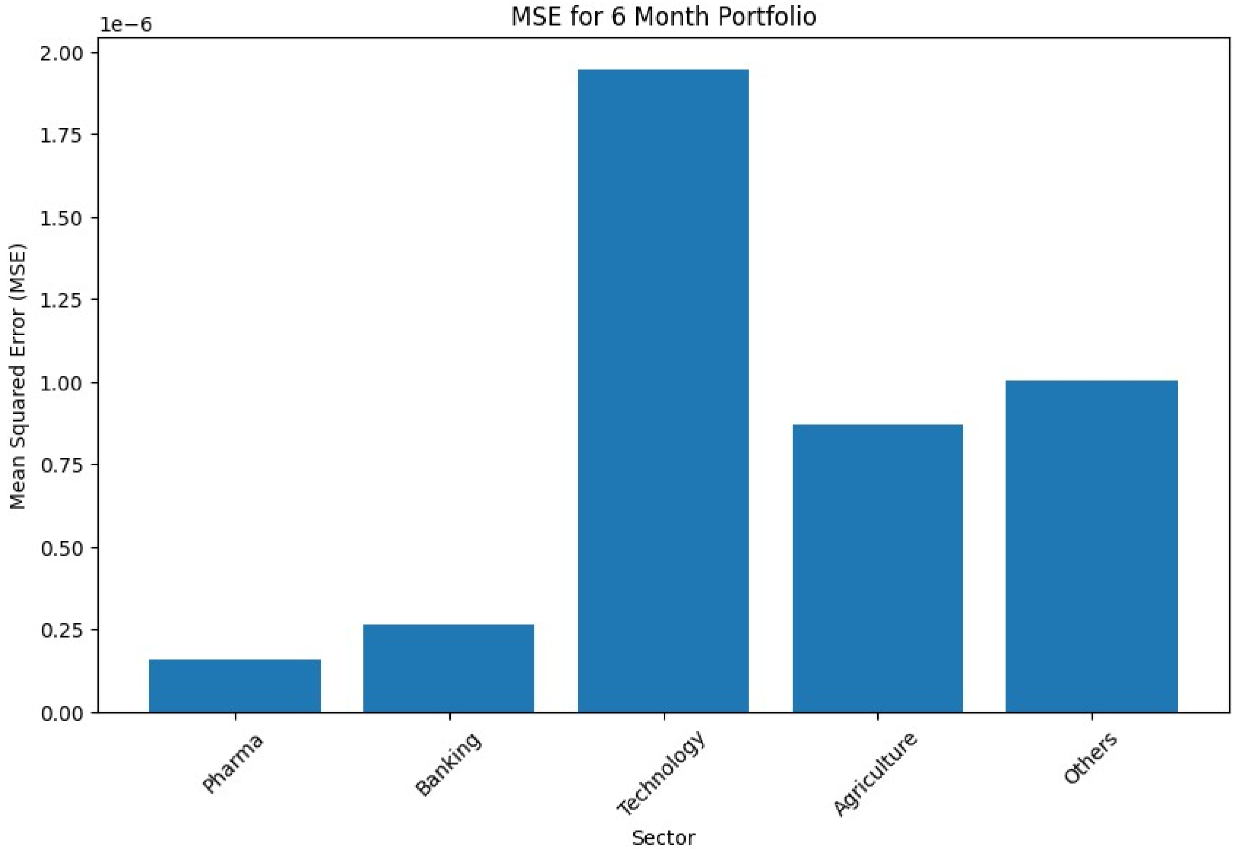
Plot mean squared error (MSE) for different time horizons

```
for month in months_to_simulate:
    plt.figure(figsize=(10, 6))
    plt.bar(data.keys(), results[month])
    plt.xlabel("Sector")
    plt.ylabel("Mean Squared Error (MSE)")
    plt.title(f"MSE for {month} Month Portfolio")
    plt.xticks(rotation=45)
    plt.show()
```



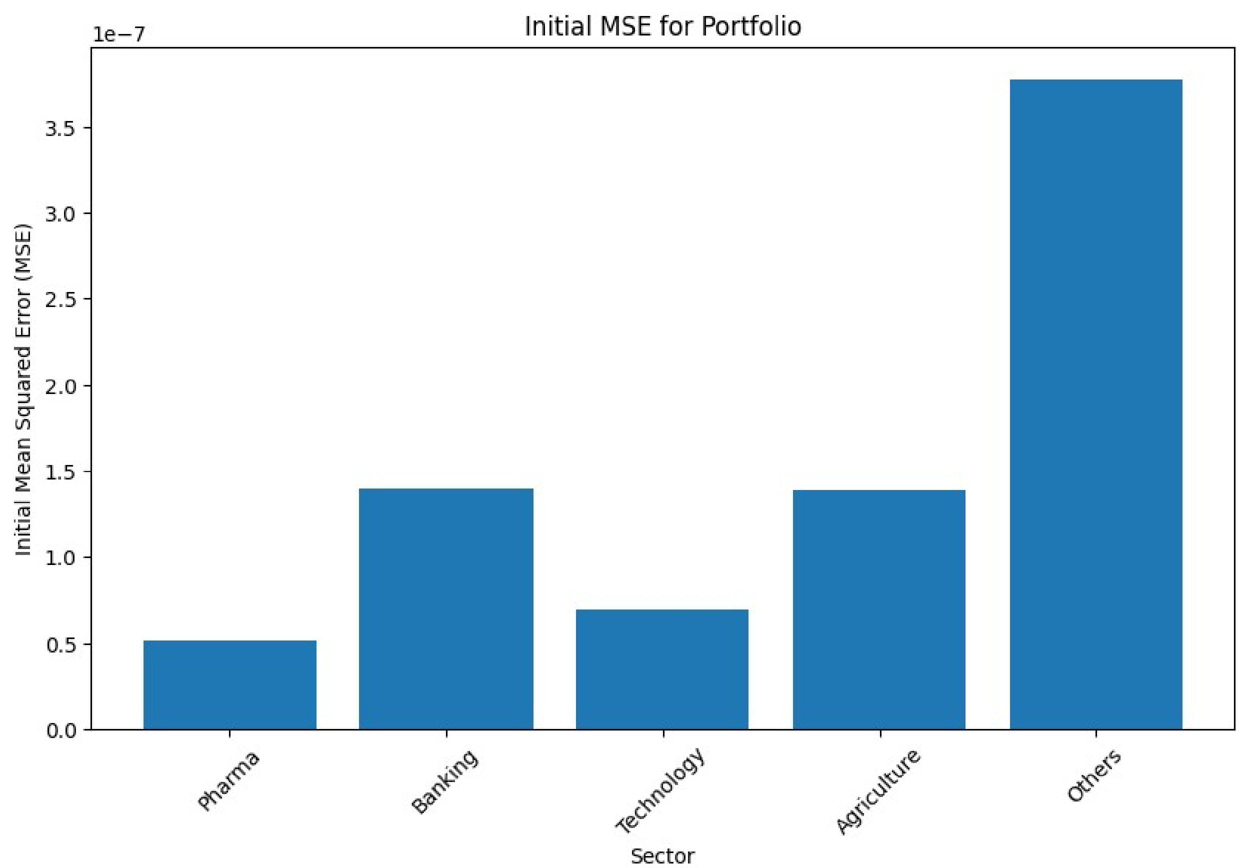
MSE for 1 Month Portfolio
1e−6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Mean Squared Error (MSE)
Pharma
Banking
Technology
Agriculture
Others
Sector


MSE for 3 Month Portfolio
1e−6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Mean Squared Error (MSE)
Pharma
Banking
Technology
Agriculture
Others
Sector



Plot initial MSE

```
plt.figure(figsize=(10, 6))
plt.bar(data.keys(), initial_mse.values())
plt.xlabel("Sector")
plt.ylabel("Initial Mean Squared Error (MSE)")
plt.title("Initial MSE for Portfolio")
plt.xticks(rotation=45)
plt.show()
```



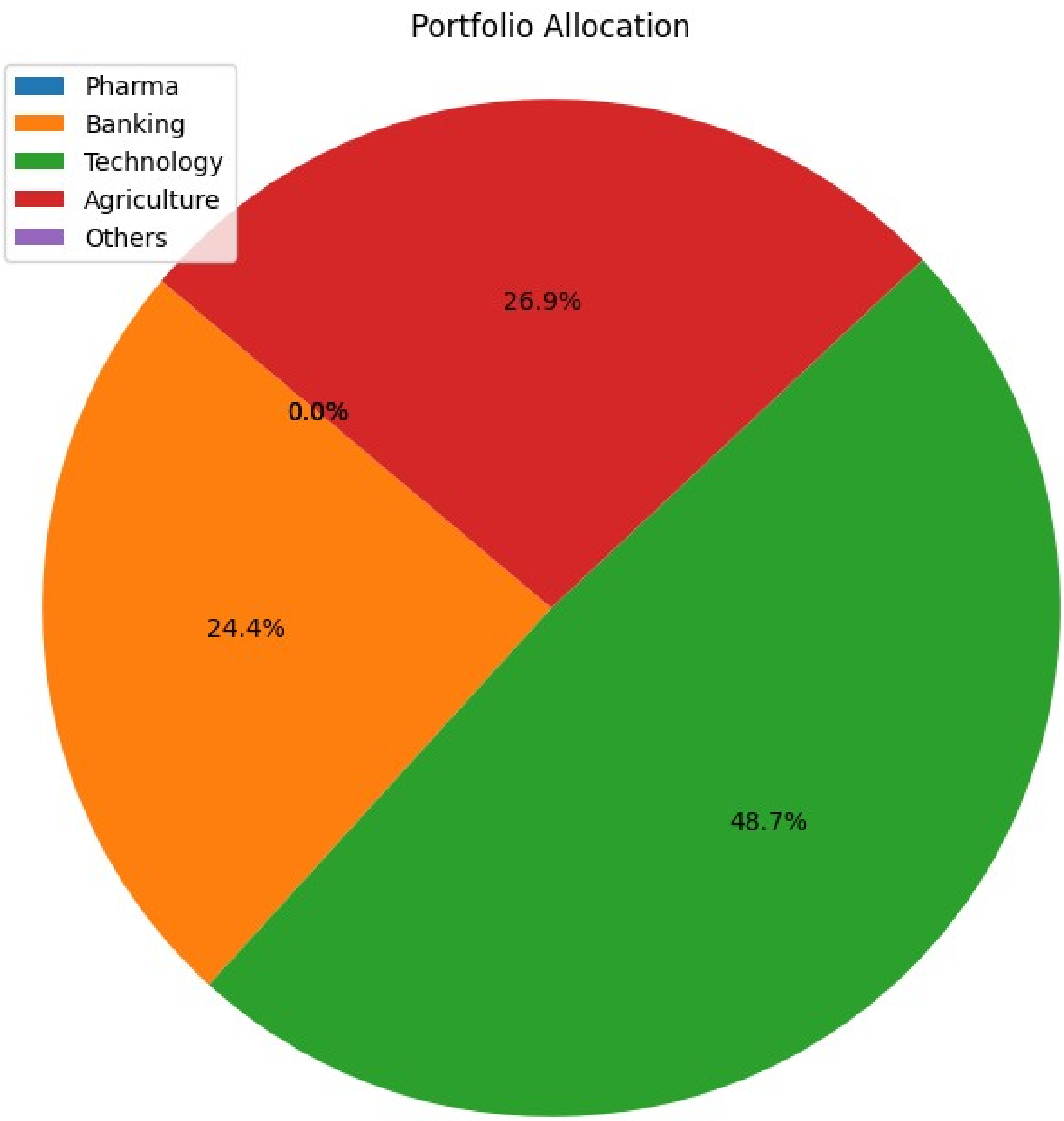

```
# Calculate the sum of portfolio weights for each sector
sector_weights = {}
for sector, weights in w_b.items():
    # Ensure that weights are non-negative (set negative weights to
zero)
    weights = np.maximum(weights, 0)
    # Recalculate the sum after setting negative weights to zero
    total_weight = sum(weights)
    if total_weight > 0:
        # Normalize weights to maintain allocation proportions
        weights = weights / total_weight
    sector_weights[sector] = total_weight

# Create a pie chart to visualize portfolio allocation
plt.figure(figsize=(8, 8))
plt.pie(sector_weights.values(), labels=None, autopct='%1.1f%%',
startangle=140)
plt.title('Portfolio Allocation')

# Add a legend to represent the sectors
plt.legend(sector_weights.keys(), loc='best')

plt.axis('equal')  # Equal aspect ratio ensures that the pie chart is
```

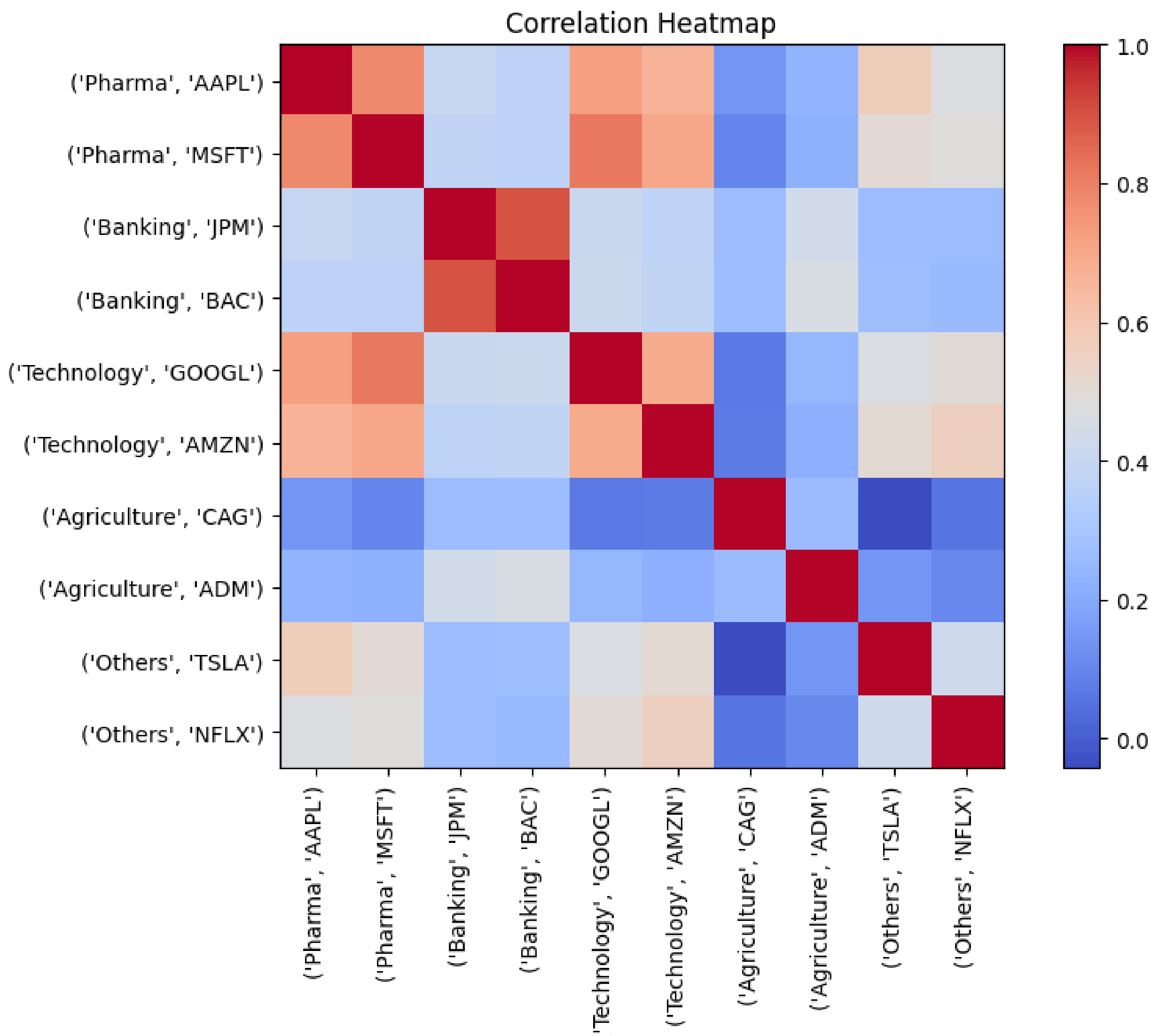
```
drawn as a circle.
plt.show()
```



Creating a heatmap to visualize the correlation between daily returns of different sectors in your portfolio.

```
correlation_matrix = pd.concat(data, axis=1).corr()

plt.figure(figsize=(10, 6))
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='none')
plt.colorbar()
plt.xticks(range(len(correlation_matrix)), correlation_matrix.columns,
rotation=90)
plt.yticks(range(len(correlation_matrix)), correlation_matrix.columns)
plt.title('Correlation Heatmap')
plt.show()
```



Correlation Heatmap
('Pharma', 'AAPL')
('Pharma', 'MSFT')
('Banking', 'JPM')
('Banking', 'BAC')
('Technology', 'GOOGL')
('Technology', 'AMZN')
('Agriculture', 'CAG')
('Agriculture', 'ADM')
('Others', 'TSLA')
('Others', 'NFLX')
1.0
0.8
0.6
0.4
0.2
0.0