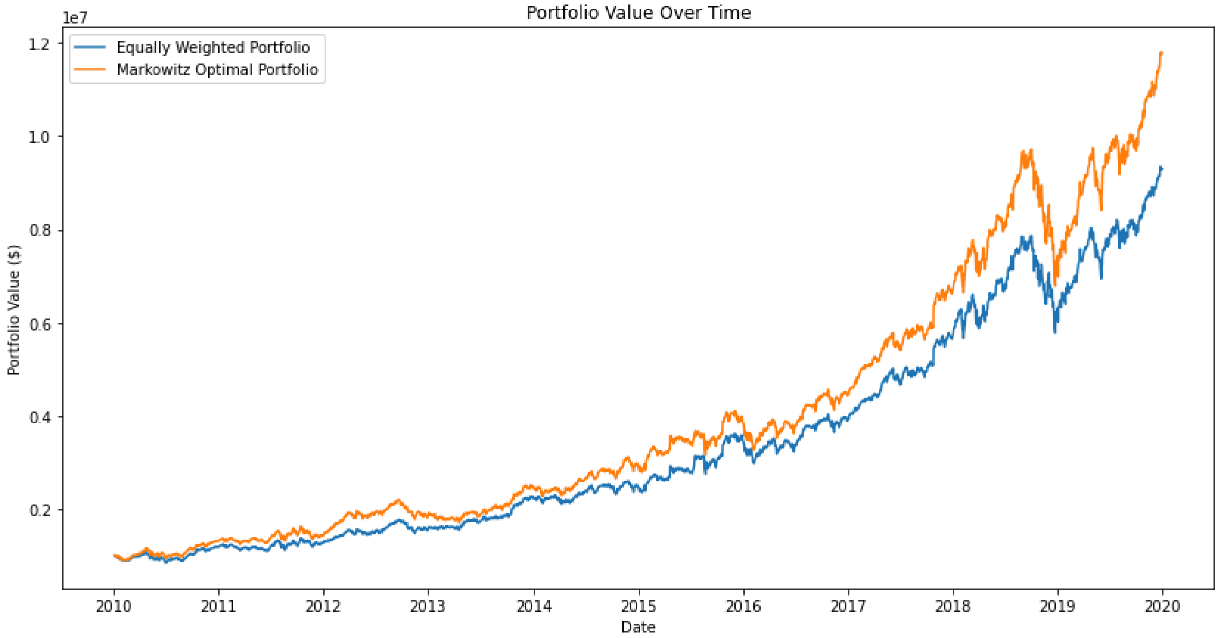
In [1]:

```
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# Download historical data
tickers = ['AAPL', 'MSFT', 'GOOGL', 'AMZN']
data = yf.download(tickers, start="2010-01-01", end="2020-01-01")['Adj Close']

# Calculate daily returns
returns = data.pct_change().dropna()

# Calculate mean returns and covariance matrix
mean_returns = returns.mean()
cov_matrix = returns.cov()

# Equally weighted portfolio
weights_equal = np.array([0.25, 0.25, 0.25, 0.25])

# Expected return, variance, and volatility for equally weighted portfolio
portfolio_return_equal = np.dot(weights_equal, mean_returns)
portfolio_variance_equal = np.dot(weights_equal.T, np.dot(cov_matrix, weights_equal)
portfolio_volatility_equal = np.sqrt(portfolio_variance_equal)

# Simulate the equally weighted portfolio
initial_investment = 1000000  # $1,000,000
portfolio_daily_returns_equal = returns.dot(weights_equal)
portfolio_value_equal = (1 + portfolio_daily_returns_equal).cumprod() * initial_inve

# Optimize portfolio using Markowitz theory
def portfolio_performance(weights, mean_returns, cov_matrix):
    returns = np.dot(weights, mean_returns)
    volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    return returns, volatility

def negative_sharpe_ratio(weights, mean_returns, cov_matrix, risk_free_rate=0):
    p_returns, p_volatility = portfolio_performance(weights, mean_returns, cov_matri
    return -(p_returns - risk_free_rate) / p_volatility

num_assets = len(tickers)
args = (mean_returns, cov_matrix)
constraints = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1})
bound = (0.0, 1.0)
bounds = tuple(bound for asset in range(num_assets))

optimal_portfolio = minimize(negative_sharpe_ratio, num_assets*[1./num_assets,], arg
weights_optimal = optimal_portfolio.x

# Expected return, variance, and volatility for optimal portfolio
portfolio_return_optimal, portfolio_volatility_optimal = portfolio_performance(weigh

# Simulate the optimal portfolio
portfolio_daily_returns_optimal = returns.dot(weights_optimal)
portfolio_value_optimal = (1 + portfolio_daily_returns_optimal).cumprod() * initial_

# Performance metrics
def calculate_metrics(portfolio_value):
    cumulative_return = portfolio_value[-1] / portfolio_value[0] - 1
    annualized_return = (1 + cumulative_return) ** (252 / len(portfolio_daily_return
    annualized_volatility = portfolio_volatility_equal * np.sqrt(252)
    sharpe_ratio = annualized_return / annualized_volatility
```

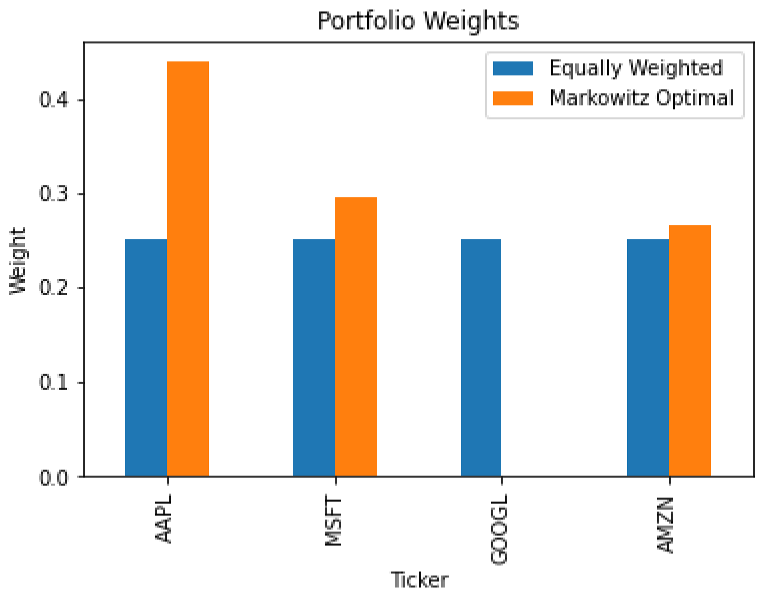
```
    return cumulative_return, annualized_return, annualized_volatility, sharpe_ratio

metrics_equal = calculate_metrics(portfolio_value_equal)
metrics_optimal = calculate_metrics(portfolio_value_optimal)

# Print performance metrics
print(f"Equally Weighted Portfolio:")
print(f"Cumulative Return: {metrics_equal[0]:.2%}")
print(f"Annualized Return: {metrics_equal[1]:.2%}")
print(f"Annualized Volatility: {metrics_equal[2]:.2%}")
print(f"Sharpe Ratio: {metrics_equal[3]:.2f}\n")

print(f"Markowitz Optimal Portfolio:")
print(f"Cumulative Return: {metrics_optimal[0]:.2%}")
print(f"Annualized Return: {metrics_optimal[1]:.2%}")
print(f"Annualized Volatility: {metrics_optimal[2]:.2%}")
print(f"Sharpe Ratio: {metrics_optimal[3]:.2f}\n")

# Plot portfolio values
plt.figure(figsize=(14, 7))
plt.plot(portfolio_value_equal, label='Equally Weighted Portfolio')
plt.plot(portfolio_value_optimal, label='Markowitz Optimal Portfolio')
plt.title('Portfolio Value Over Time')
plt.xlabel('Date')
plt.ylabel('Portfolio Value ($)')
plt.legend()
plt.show()

# Display portfolio weights
weights_df = pd.DataFrame({
    'Ticker': tickers,
    'Equally Weighted': weights_equal,
    'Markowitz Optimal': weights_optimal
})

print(weights_df)

# Plot portfolio weights
plt.figure(figsize=(14, 7))
weights_df.set_index('Ticker').plot(kind='bar')
plt.title('Portfolio Weights')
plt.xlabel('Ticker')
plt.ylabel('Weight')
plt.show()

# Generate portfolios for Efficient Frontier
num_portfolios = 10000
results = np.zeros((3, num_portfolios))
for i in range(num_portfolios):
    weights = np.random.random(num_assets)
    weights /= np.sum(weights)
    portfolio_return, portfolio_volatility = portfolio_performance(weights, mean_ret
    results[0,i] = portfolio_volatility
    results[1,i] = portfolio_return
    results[2,i] = (portfolio_return - 0) / portfolio_volatility  # Assuming risk-fr

# Plot Efficient Frontier
plt.figure(figsize=(14, 7))
plt.scatter(results[0,:], results[1,:], c=results[2,:], cmap='YlGnBu', marker='o')
plt.title('Efficient Frontier')
plt.xlabel('Volatility')
plt.ylabel('Return')
plt.colorbar(label='Sharpe Ratio')
```

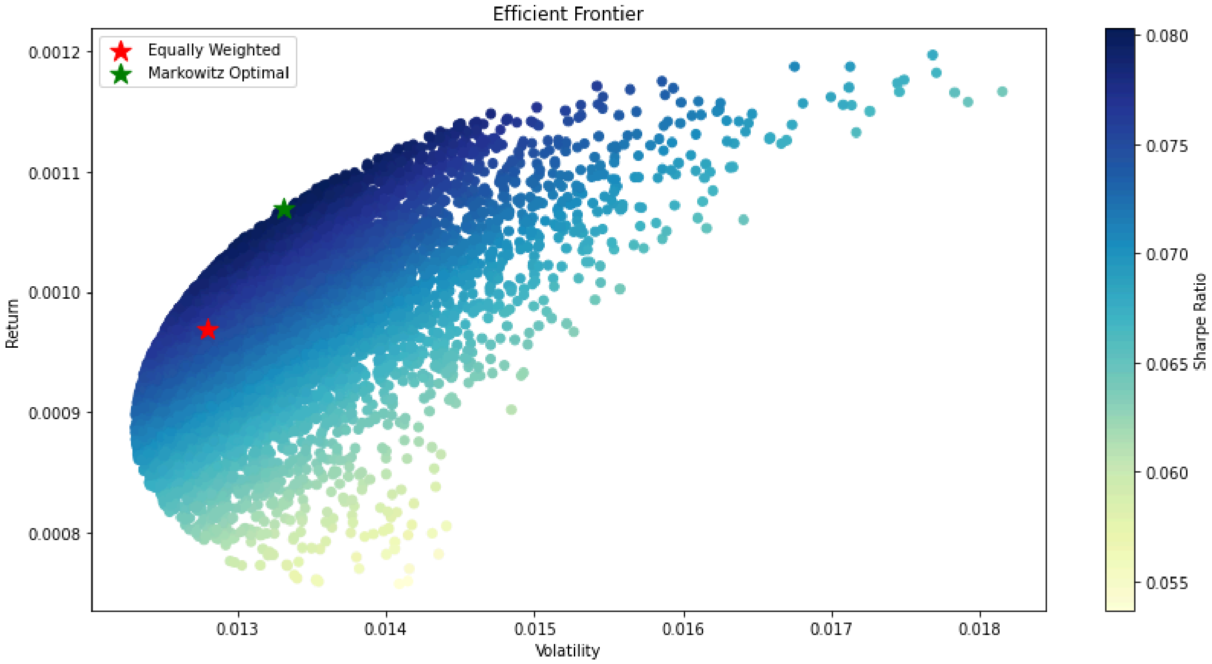
```
# Highlight Equally Weighted and Optimal Portfolios
plt.scatter(portfolio_volatility_equal, portfolio_return_equal, marker='*', color='r
plt.scatter(portfolio_volatility_optimal, portfolio_return_optimal, marker='*', colo
plt.legend()
plt.show()
```

```
[*********************100%***********************]  4 of 4 completed
Equally Weighted Portfolio:
Cumulative Return: 820.93%
Annualized Return: 24.90%
Annualized Volatility: 20.31%
Sharpe Ratio: 1.23

Markowitz Optimal Portfolio:
Cumulative Return: 1068.07%
Annualized Return: 27.91%
Annualized Volatility: 20.31%
Sharpe Ratio: 1.37
```




```
  Ticker  Equally Weighted  Markowitz Optimal
0   AAPL              0.25           0.439250
1   MSFT              0.25           0.295837
2  GOOGL              0.25           0.000000
3   AMZN              0.25           0.264913
<Figure size 1008x504 with 0 Axes>
```




Efficient Frontier
Equally Weighted
Markowitz Optimal
Return
Volatility
Sharpe Ratio
0.0012
0.0011
0.0010
0.0009
0.0008
0.013
0.014
0.015
0.016
0.017
0.018
0.080
0.075
0.070
0.065
0.060
0.055

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: