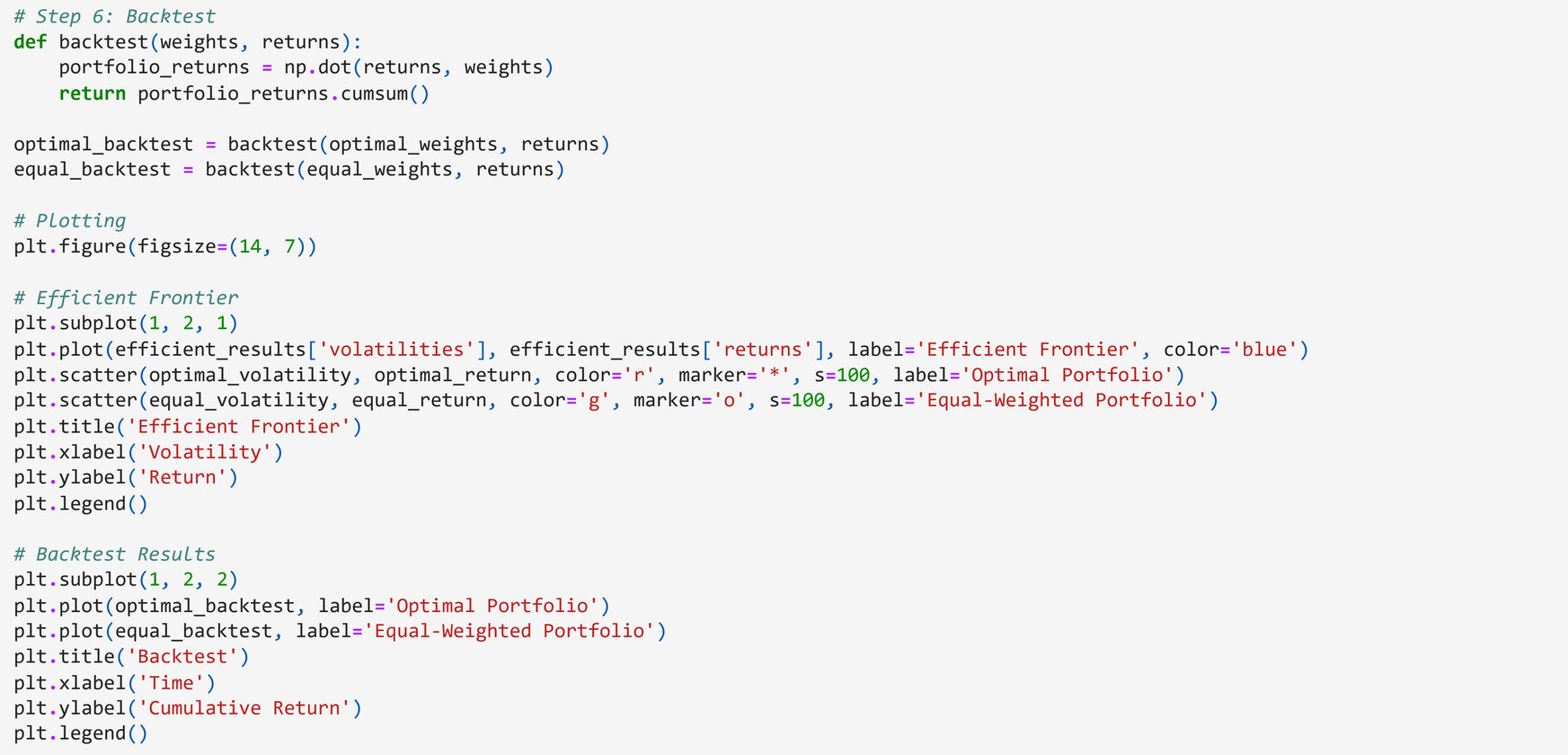
In [8]:

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

# Step 1: Download stock data
symbols = ['MSFT', 'AAPL', 'SBUX', 'TSLA', 'MCD']
start_date = '2018-01-01'
end_date = '2024-07-31'

data = yf.download(symbols, start=start_date, end=end_date)['Close']

# Step 2: Calculate returns and covariance matrix
returns = data.pct_change().dropna()
mean_returns = returns.mean()
cov_matrix = returns.cov()

# Define functions for portfolio performance
def portfolio_performance(weights, mean_returns, cov_matrix):
    portfolio_return = np.dot(weights, mean_returns)
    portfolio_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    return portfolio_return, portfolio_volatility

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

def get_optimal_weights(mean_returns, cov_matrix, risk_free_rate):
    num_assets = len(mean_returns)
    args = (mean_returns, cov_matrix, risk_free_rate)
    constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
    bounds = tuple((0, 1) for _ in range(num_assets))
    result = minimize(negative_sharpe_ratio, num_assets * [1. / num_assets,], args=args,
                      method='SLSQP', bounds=bounds, constraints=constraints)
    return result.x

# Step 3: Calculate Markowitz optimal portfolio
risk_free_rate = 0.01
optimal_weights = get_optimal_weights(mean_returns, cov_matrix, risk_free_rate)
optimal_return, optimal_volatility = portfolio_performance(optimal_weights, mean_returns, cov_matrix)

# Step 4: Calculate equally-weighted portfolio
num_assets = len(symbols)
equal_weights = np.array([1. / num_assets] * num_assets)
equal_return, equal_volatility = portfolio_performance(equal_weights, mean_returns, cov_matrix)

# Step 5: Efficient Frontier
def portfolio_variance(weights, cov_matrix):
    return np.dot(weights.T, np.dot(cov_matrix, weights))

def efficient_frontier(mean_returns, cov_matrix, returns_range):
    num_assets = len(mean_returns)
    bounds = tuple((0, 1) for _ in range(num_assets))
    results = {'returns': [], 'volatilities': [], 'weights': []}
    for r in returns_range:
        constraints = ({'type': 'eq', 'fun': lambda x: np.dot(x, mean_returns) - r},
                       {'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
        result = minimize(portfolio_variance, num_assets * [1. / num_assets,], args=(cov_matrix,),
                          method='SLSQP', bounds=bounds, constraints=constraints)
        results['returns'].append(r)
        results['volatilities'].append(np.sqrt(result.fun))
        results['weights'].append(result.x)
    return results

# Calculate returns range for the efficient frontier
returns_range = np.linspace(mean_returns.min(), mean_returns.max(), 100)
efficient_results = efficient_frontier(mean_returns, cov_matrix, returns_range)

# Step 6: Backtest
def backtest(weights, returns):
    portfolio_returns = np.dot(returns, weights)
    return portfolio_returns.cumsum()

optimal_backtest = backtest(optimal_weights, returns)
equal_backtest = backtest(equal_weights, returns)

# Plotting
plt.figure(figsize=(14, 7))

# Efficient Frontier
plt.subplot(1, 2, 1)
plt.plot(efficient_results['volatilities'], efficient_results['returns'], label='Efficient Frontier', color='blue')
plt.scatter(optimal_volatility, optimal_return, color='r', marker='*', s=100, label='Optimal Portfolio')
plt.scatter(equal_volatility, equal_return, color='g', marker='o', s=100, label='Equal-Weighted Portfolio')
plt.title('Efficient Frontier')
plt.xlabel('Volatility')
plt.ylabel('Return')
plt.legend()

# Backtest Results
plt.subplot(1, 2, 2)
plt.plot(optimal_backtest, label='Optimal Portfolio')
plt.plot(equal_backtest, label='Equal-Weighted Portfolio')
plt.title('Backtest')
plt.xlabel('Time')
plt.ylabel('Cumulative Return')
plt.legend()

plt.tight_layout()
plt.show()
```

```
[**********100%**********]  5 of 5 completed
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: