In [89]:
```
pip install yfinance
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: yfinance in /usr/local/lib/python3.7/dist-packages (0.1.74)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dist-packages (from yfinance)
(0.0.11)
Requirement already satisfied: requests>=2.26 in /usr/local/lib/python3.7/dist-packages (from yfinance) (2.28.
1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.3.5)
Requirement already satisfied: lxml>=4.5.1 in /usr/local/lib/python3.7/dist-packages (from yfinance) (4.9.1)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.21.6)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=
0.24.0->yfinance) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.0->yfi
nance) (2022.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3-
>pandas>=0.24.0->yfinance) (1.15.0)
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.7/dist-packages (from request
s>=2.26->yfinance) (2.1.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfi
nance) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests>=
2.26->yfinance) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.2
6->yfinance) (2022.6.15)
```

In [90]:
```
# importing all the necessary modules
import pandas as pd
import yfinance as yf
import datetime as dt
import numpy as np
import seaborn as sns
import scipy.optimize as sco

import matplotlib.pyplot as plt
```

In [91]:
```
# Choose and store seven (7) assets into a list, called tickers
tickers=['AAPL', 'GE', 'MSFT', 'MCD', 'JPM', 'SBUX', 'TSLA']

# Retrieve daily ajusted close data on the seven assets for the previous 2 years
df= yf.download(tickers, period = '2y')['Adj Close']
```

```
[*********************100%***********************]  7 of 7 completed
```

In [92]:

```
# New column name called portfolio and set it equal to the average adj close across row.
df['portfolio'] = df.mean(axis=1)
returns= df.pct_change()
```

In [93]:

```
df.head()
```

Out[93]:

Date	AAPL	GE	JPM	MCD	MSFT	SBUX	TSLA	portfolio
2020-08-06	112.497429	50.245903	91.925209	194.147980	212.539764	73.191086	297.915985	147.494765
2020-08-07	109.939713	50.801556	93.948257	195.504883	208.737930	73.316841	290.541992	146.113025
2020-08-10	111.537666	52.944740	95.139389	195.046234	204.582413	74.942024	283.713989	145.415208
2020-08-11	108.220558	53.421001	98.145576	195.887115	199.798187	76.296341	274.877991	143.806681
2020-08-12	111.817184	53.341629	97.313683	196.861755	205.505875	76.702629	310.951996	150.356393

In [94]:

```
returns.head()
```

Out[94]:

Date	AAPL	GE	JPM	MCD	MSFT	SBUX	TSLA	portfolio
2020-08-06	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2020-08-07	-0.022736	0.011059	0.022008	0.006989	-0.017888	0.001718	-0.024752	-0.009368
2020-08-10	0.014535	0.042187	0.012679	-0.002346	-0.019908	0.022167	-0.023501	-0.004776
2020-08-11	-0.029740	0.008995	0.031598	0.004311	-0.023385	0.018072	-0.031144	-0.011062
2020-08-12	0.033234	-0.001486	-0.008476	0.004976	0.028567	0.005325	0.131236	0.045545

In [95]:

```
ticker_returns=returns.iloc[:,:-1]
port_returns=returns.iloc[:,-1]
print(returns.shape, ticker_returns.shape, port_returns.shape)
```

```
(504, 8) (504, 7) (504,)
```

In [96]:
```
returns.iloc[-251:].mean() *252      # returns.mean() * (252*2)
```

Out[96]:
```
AAPL         0.174552
GE          -0.281500
JPM         -0.244430
MCD          0.131999
MSFT         0.030637
SBUX        -0.256194
TSLA         0.414241
portfolio    0.132549
dtype: float64
```

In [97]:
```
returns.iloc[-251:].var() *252       # returns.var() * (252*2)
```

Out[97]:
```
AAPL         0.092882
GE           0.111850
JPM          0.072743
MCD          0.033437
MSFT         0.091178
SBUX         0.106715
TSLA         0.401953
portfolio    0.140547
dtype: float64
```

In [98]:
```
returns.corr()  # correlation matrix for the last two year
# returns.iloc[-251:].corr()  correlation matrix for the last year
```

Out[98]:

	AAPL	GE	JPM	MCD	MSFT	SBUX	TSLA	portfolio
AAPL	1.000000	0.201129	0.243622	0.396652	0.738105	0.489914	0.556500	0.691213
GE	0.201129	1.000000	0.561077	0.310950	0.154754	0.375387	0.175077	0.315647
JPM	0.243622	0.561077	1.000000	0.333713	0.231095	0.434404	0.178033	0.336840
MCD	0.396652	0.310950	0.333713	1.000000	0.407925	0.558009	0.208568	0.384630
MSFT	0.738105	0.154754	0.231095	0.407925	1.000000	0.504126	0.510061	0.666360
SBUX	0.489914	0.375387	0.434404	0.558009	0.504126	1.000000	0.351464	0.524436
TSLA	0.556500	0.175077	0.178033	0.208568	0.510061	0.351464	1.000000	0.955140

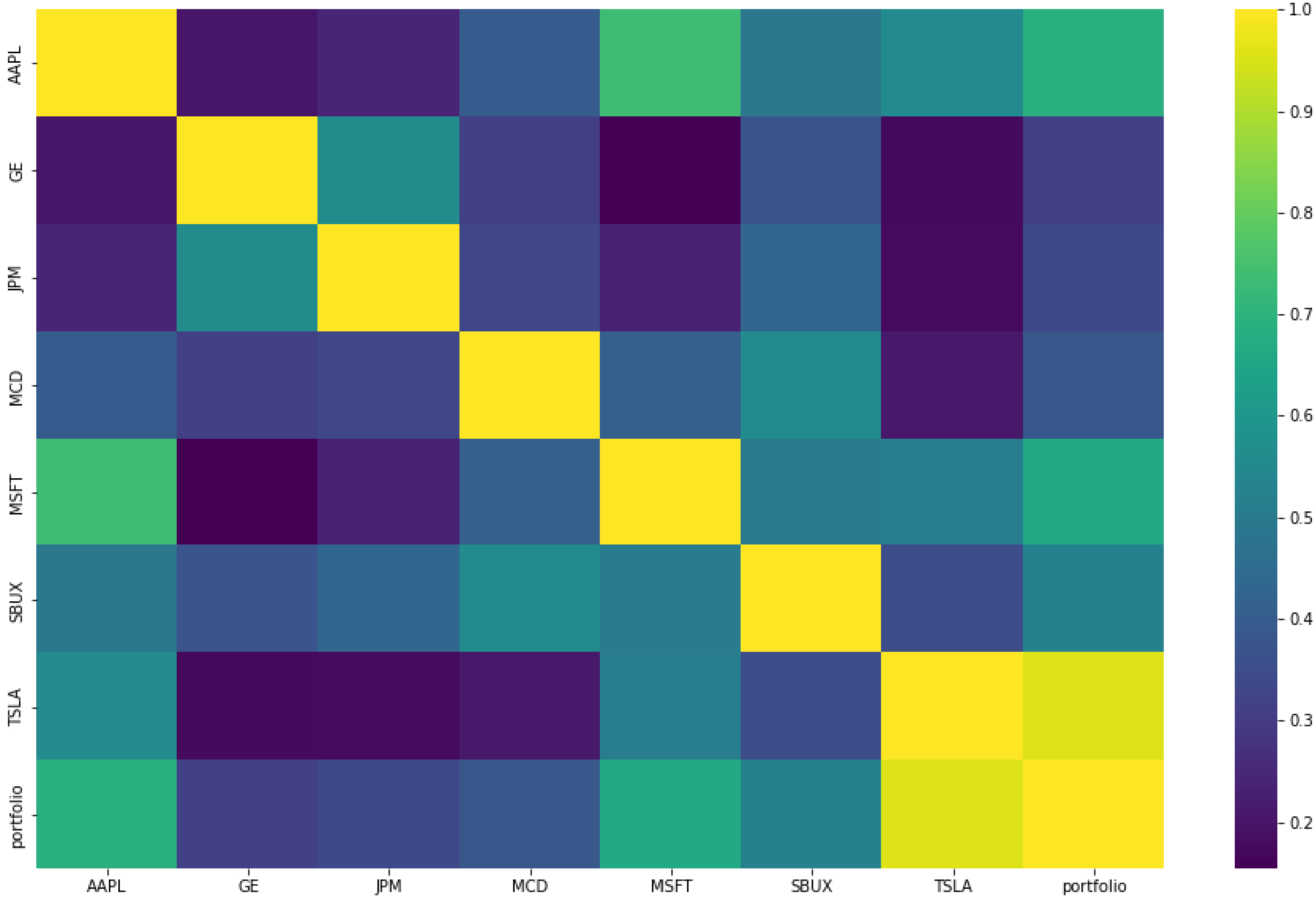
	AAPL	GE	JPM	MCD	MSFT	SBUX	TSLA	portfolio
portfolio	0.691213	0.315647	0.336840	0.384630	0.666360	0.524436	0.955140	1.000000

In [99]:

```
fig, ax = plt.subplots(figsize=(16, 10))
sns.heatmap(returns.corr(), cmap = 'viridis')
```

Out[99]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1993feca50>
```



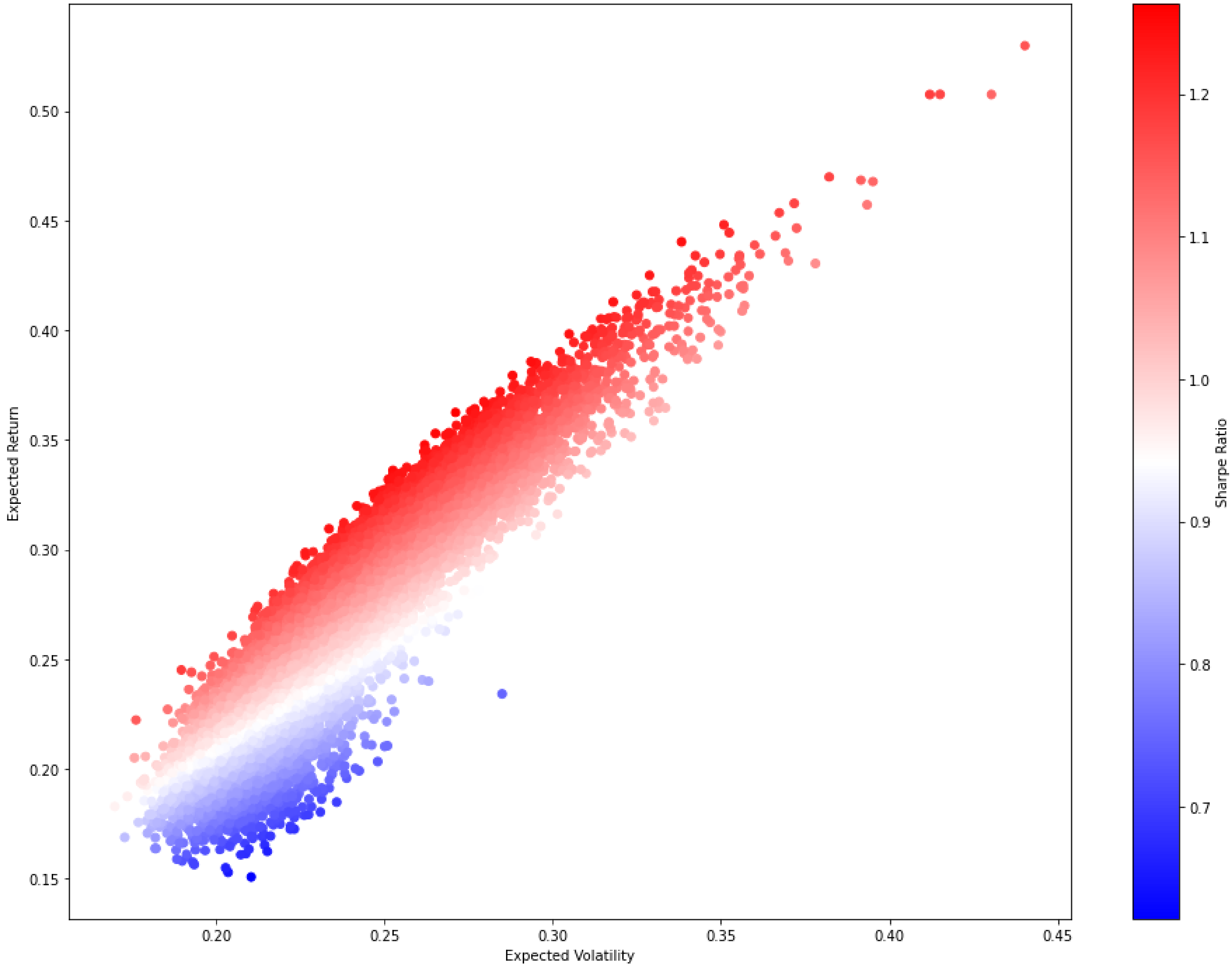
In [100…

```
# Monte Carlo calulation using class
class Monte_carlo:
  exp_returns = []
  exp_volatility = []



  def __init__(self, ticker_returns):
```

```
        self.ticker_returns = ticker_returns

# Annualized portfolio return given the portfolio weights
    def port_ret(self, weights):
        return np.sum(self.ticker_returns.mean() * weights) * 252

# Annualized portfolio volatility given the portfolio weights
    def port_vol(self, weights):
        return np.sqrt(np.dot(weights.T, np.dot(self.ticker_returns.cov() * 252, weights)))

    def mc_simulation(self, tickers):
# Monte Carlo simulation of portfolio weights: randomize weight to find the best possible combination
        weight_app =[]
        for run in range (20000):
            weights = np.random.random(len(tickers))
            weights /= np.sum(weights)
            weight_app.append(weights)
# Append the results into list objects
            # self.weights.append(weights)
            self.exp_returns.append(self.port_ret(weights))
            self.exp_volatility.append(self.port_vol(weights))

        # weight_app =np.array(weight_app)
        exp_returns = np.array(self.exp_returns)
        exp_volatility = np.array(self.exp_volatility)
        sharpe_ratio= (exp_returns -0.02)/exp_volatility          # setting free_risk_rate =2%


        plt.figure(figsize=(16, 12))
        plt.scatter(self.exp_volatility, self.exp_returns, c=sharpe_ratio ,marker='o', cmap='bwr')
        plt.xlabel('Expected Volatility')
        plt.ylabel('Expected Return')
        plt.colorbar(label='Sharpe Ratio')
        return weight_app, exp_returns,exp_volatility,sharpe_ratio
```

In [101…

```
object1 = Monte_carlo(ticker_returns)
result = object1.mc_simulation(tickers)
```



In [102…

```
df_weight=pd.DataFrame(value , columns=['Weight'])
```

```
ex_ret = pd.DataFrame(result[1], columns=['Expected_return'])
ex_vol= pd.DataFrame(result[2],columns=['Expected_volatility'])
ex_sharpe= pd.DataFrame(result[3],columns=['Sharpe_ratio'])
```

In [103…

```
pd.concat([df_weight,ex_ret,ex_vol,ex_sharpe], axis=1).head()
```

Out[103…

	Weight	Expected_return	Expected_volatility	Sharpe_ratio
0	[0.2159033663586187, 0.01294834138096075, 0.20...	0.301060	0.250073	1.123914
1	[0.019197431656732808, 0.2032121186507859, 0.2...	0.196478	0.209437	0.842629
2	[0.1881069432974777, 0.0038799974745581454, 0....	0.273651	0.241083	1.052131
3	[0.07465283099726663, 0.19274894387441638, 0.1...	0.321320	0.273233	1.102796
4	[0.3014040170284223, 0.28252821284567137, 0.20...	0.269996	0.227475	1.099008

In [104…

```
def port_ret(weights):
    return np.sum(ticker_returns.mean() * weights) * 252

# Annualized portfolio volatility given the portfolio weights
def port_vol(weights):
    return np.sqrt(np.dot(weights.T, np.dot(ticker_returns.cov() * 252, weights)))

exp_returns = []
exp_volatility=[]
# Monte Carlo simulation of portfolio weights: randomize weight to find the best possible combination
for run in range (10000):
    weights = np.random.random(len(tickers))
    weights /= np.sum(weights)
# Append the results into list objects
    exp_returns.append(port_ret(weights))
    exp_volatility.append(port_vol(weights))
exp_returns = np.array(exp_returns)
exp_volatility = np.array(exp_volatility)
sharpe_ratio= (exp_returns -0.02)/exp_volatility          # setting free_risk_rate =2%
```

In [105…

```
# Linear Regression Calculation
# Function to be minimized
def min_func_sharpe(weights):
  return -(port_ret(weights)-0.02) / port_vol(weights)
```

```python
# Equality constraint
cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
# Bounds for the parameters
bnds = tuple((0, 1) for x in range(len(tickers)))
# Equal weights vector
eweights = np.array((len(tickers)) * [1. / (len(tickers)),])
# optimization of function min_func_sharpe()
opts = sco.minimize(min_func_sharpe, eweights,
                    method='SLSQP', bounds=bnds, constraints=cons)
# optimization or minimizing the volatility
optv = sco.minimize(port_vol, eweights,
                    method='SLSQP', bounds=bnds, constraints=cons)
# The two binding constraints for the efficient frontier
cons = ({'type': 'eq', 'fun': lambda x: port_ret(x) - tret},
 {'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
bnds = tuple((0, 1) for x in weights)
trets = np.linspace(0.15, 0.55, 50)
tvols = []
# The minimization of portfolio volatility for different target returns
for tret in trets:
  res = sco.minimize(port_vol, eweights, method='SLSQP', bounds=bnds, constraints=cons)
  tvols.append(res['fun'])
tvols = np.array(tvols)
```

In [106…

```python
# The optimal portfolio weights
opts['x'].round(5)
```

Out[106…

```
array([0.     , 0.17458, 0.     , 0.55593, 0.     , 0.     , 0.26948])
```

In [107…

```python
# The optimal portfolio weighting return
port_ret(opts['x']).round(3)
```

Out[107…

```
0.336
```

In [108…

```python
# The optimal portfolio weighting volatility
port_vol(opts['x']).round(3)
```

Out[108…

```
0.243
```

In [109…

```python
# The maximum Sharpe ratio
```

```
(port_ret(opts['x'])-0.02) / port_vol(opts['x'])
```

Out[109… 1.3024689648145047

In [112…

```
plt.figure(figsize=(10, 6))
plt.scatter(result[2], result[1], c=(result[1]-0.02)/ result[2],marker='o', alpha=0.8, cmap='coolwarm')
plt.plot(tvols, trets, 'b', lw=5.0)
plt.plot(port_vol(opts['x']), port_ret(opts['x']),'y*', markersize=20)
plt.plot(port_vol(optv['x']), port_ret(optv['x']),'ro', markersize=15)
plt.xlabel('Expected Volatility')
plt.ylabel('Expected Return')
plt.colorbar(label='Sharpe Ratio')
```

Out[112… <matplotlib.colorbar.Colorbar at 0x7f1997419210>