Monte Carlo simulation

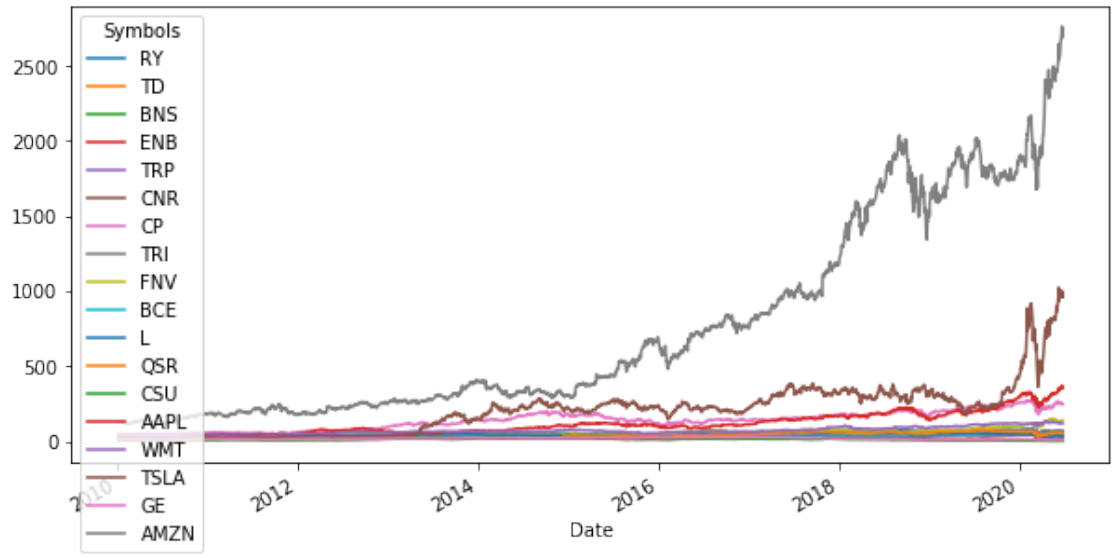
June 29, 2020

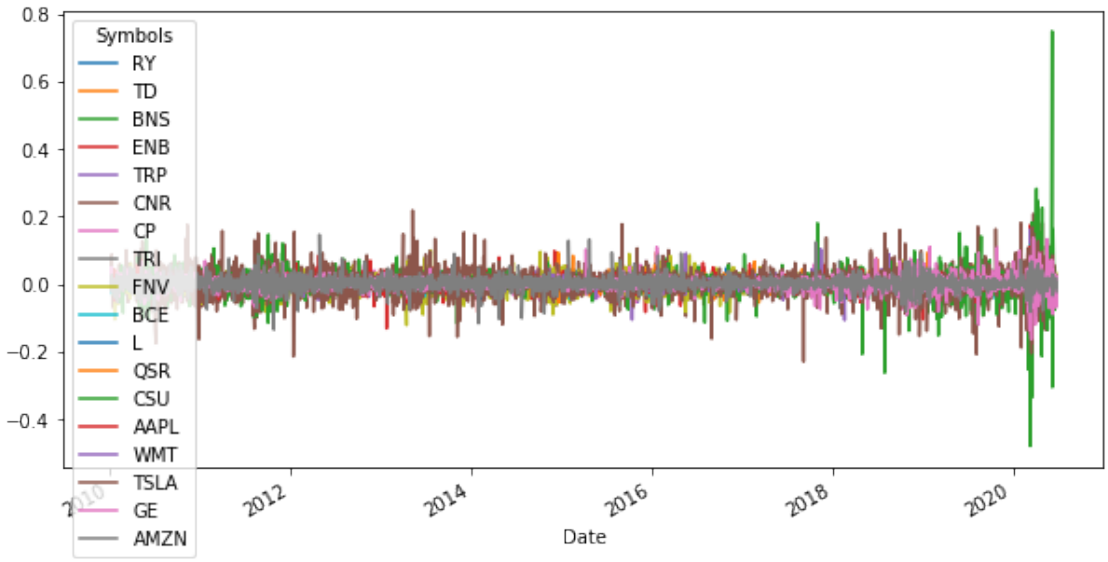
[71]:
```
import numpy as np
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import datetime
import scipy.optimize as optimization

#these are the stocks we can choose in our portfolio
stocks =␣
 ↪['RY','TD','BNS','ENB','TRP','CNR','CP','TRI','FNV','BCE','L','QSR','CSU','AAPL','WMT','TSL

#we use historical data to approximate mean and variance: MPT depends on␣
 ↪historical data !!!
start_date=datetime.datetime(2010,1,1)
end_date =datetime.datetime(2020,6,27)

#downloading the data from Yahoo! Finance
def download_data(stocks):
        data = web.
 ↪DataReader(stocks,data_source='yahoo',start=start_date,end=end_date)['Adj␣
 ↪Close']
        return data

def show_data(data):
        data.plot(figsize=(10,5))
        plt.show()

#we usually use natural logarithm for normalization purposes
def calculate_returns(data):
        returns = np.log(data/data.shift(1))
        return returns;

def plot_daily_returns(returns):
        returns.plot(figsize=(10,5))
        plt.show()
```

```
#print out mean and covariance of stocks within [start_date, end_date]. There␣
 ↪are 252 trading days within a year
def show_statistics(returns):
        print(returns.mean()*252)
        print(returns.cov()*252)

#weights defines what stocks to include (with what portion) in the portfolio
def initialize_weights():
        weights = np.random.random(len(stocks))
        weights /= np.sum(weights)
        return weights;

#expected portfolio return
def calculate_portfolio_return(returns, weights):
        portfolio_return = np.sum(returns.mean()*weights)*252
        print("Expected portfolio return:", portfolio_return)

#expected portfolio variance
def calculate_portfolio_variance(returns, weights):
        portfolio_variance = np.sqrt(np.dot(weights.T, np.dot(returns.
 ↪cov()*252,weights)))
        print("Expected variance:", portfolio_variance)

def generate_portfolios(weights, returns):

        preturns = []
        pvariances = []

        #Monte-Carlo simulation: we generate several random weights -> so␣
 ↪random portfolios !!!
        for i in range(10000):
                weights = np.random.random(len(stocks))
                weights/=np.sum(weights)
                preturns.append(np.sum(returns.mean()*weights)*252)
                pvariances.append(np.sqrt(np.dot(weights.T,np.dot(returns.
 ↪cov()*252,weights))))

        preturns = np.array(preturns)
        pvariances = np.array(pvariances)
        return preturns,pvariances

def plot_portfolios(returns, variances):
        plt.figure(figsize=(10,6))
        plt.scatter(variances,returns,c=returns/variances,marker='o')
        plt.grid(True)
        plt.xlabel('Expected Volatility')
        plt.ylabel('Expected Return')
```

```
        plt.colorbar(label='Sharpe Ratio')
        plt.show()

# OK this is the result of the simulation ... we have to find the optimal␣
 ↪portfolio with
# some optimization technique !!! scipy can optimize functions (minimum/maximum␣
 ↪finding)
def statistics(weights, returns):
        portfolio_return=np.sum(returns.mean()*weights)*252
        portfolio_volatility=np.sqrt(np.dot(weights.T,np.dot(returns.
 ↪cov()*252,weights)))
        return np.array([portfolio_return,portfolio_volatility,portfolio_return/
 ↪portfolio_volatility])

# [2] means that we want to maximize according to the Sharpe-ration
# note: maximizing f(x) function is the same as minimizing -f(x) !!!
def        min_func_sharpe(weights,returns):
        return        -statistics(weights,returns)[2]

# what are the constraints? The sum of weights = 1 !!!  f(x)=0 this is the␣
 ↪function to minimize
def optimize_portfolio(weights,returns):
        constraints = ({'type':'eq','fun': lambda x: np.sum(x)-1}) #the sum of␣
 ↪weights is 1
        bounds = tuple((0,1) for x in range(len(stocks))) #the weights can be 1␣
 ↪at most: 1 when 100% of money is invested into a single stock
        optimum=optimization.
 ↪minimize(fun=min_func_sharpe,x0=weights,args=returns,method='SLSQP',bounds=bounds,constrain
        return optimum

# optimal portfolio according to weights: 0 means no shares of that given␣
 ↪company
def print_optimal_portfolio(optimum, returns):
        print("Optimal weights:", optimum['x'].round(3))
        print("Expected return, volatility and Sharpe ratio:",␣
 ↪statistics(optimum['x'].round(3),returns))

def show_optimal_portfolio(optimum, returns, preturns, pvariances):
        plt.figure(figsize=(10,6))
        plt.scatter(pvariances,preturns,c=preturns/pvariances,marker='o')
        plt.grid(True)
        plt.xlabel('Expected Volatility')
        plt.ylabel('Expected Return')
        plt.colorbar(label='Sharpe Ratio')
```

```
        plt.
↪plot(statistics(optimum['x'],returns)[1],statistics(optimum['x'],returns)[0],'g*',markersiz
↪0)
        plt.show()

if __name__ == "__main__":
        data = download_data(stocks)
        show_data(data)
        returns = calculate_returns(data)
        plot_daily_returns(returns)
        show_statistics(returns)
        weights=initialize_weights()
        calculate_portfolio_return(returns,weights)
        calculate_portfolio_variance(returns,weights)
        preturns,pvariances=generate_portfolios(weights, returns)
        plot_portfolios(preturns,pvariances)
        optimum=optimize_portfolio(weights,returns)
        print_optimal_portfolio(optimum, returns)
        show_optimal_portfolio(optimum, returns, preturns, pvariances)
```



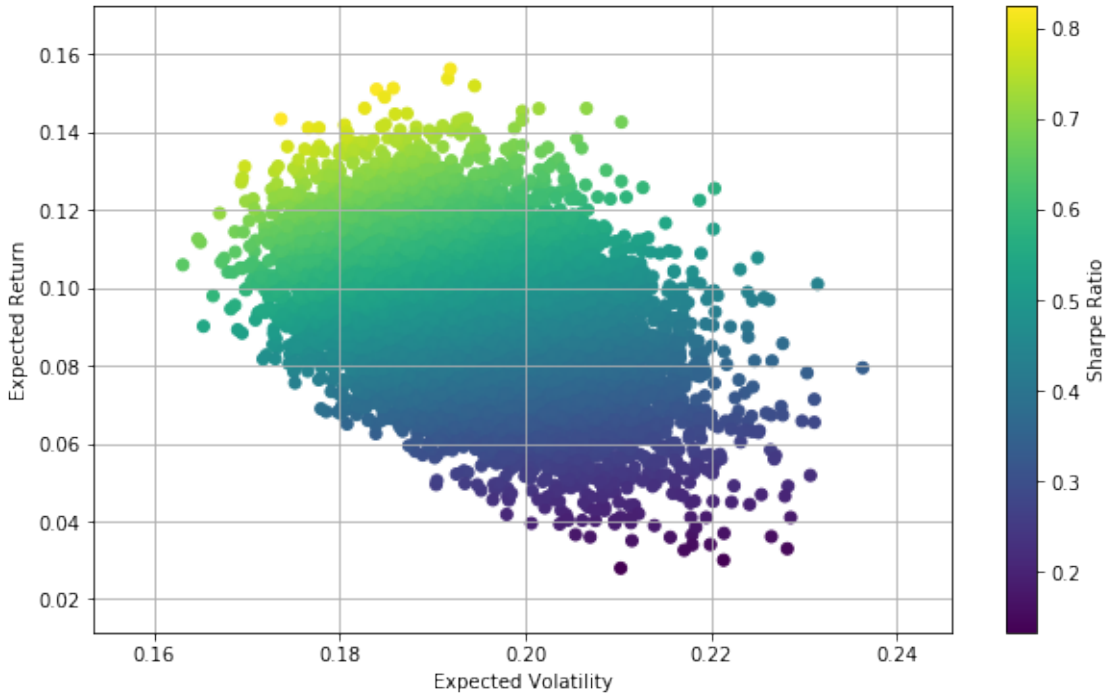



```
Symbols
RY      0.065936
TD      0.088125
BNS     0.036742
ENB     0.077957
TRP     0.071246
CNR    -0.057280
CP      0.159347
TRI     0.102345
FNV     0.168402
BCE     0.095454
L      -0.005871
QSR     0.087251
CSU    -0.186071
AAPL    0.247640
WMT     0.099571
TSLA    0.369907
GE     -0.048204
AMZN    0.286702
dtype: float64
```

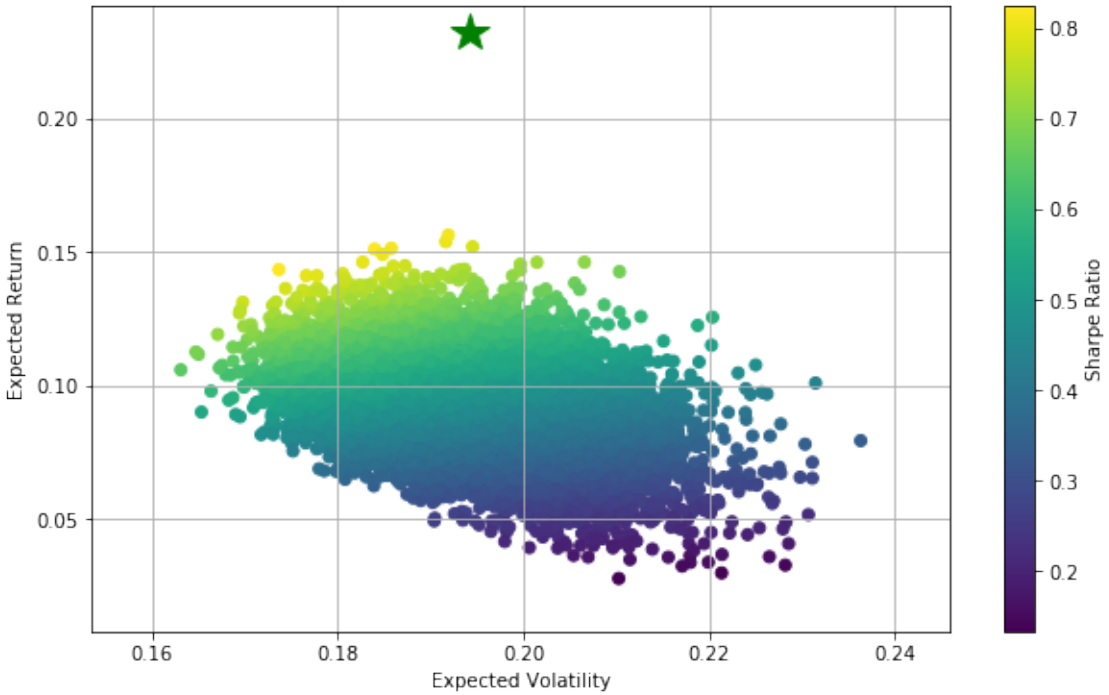
Symbols	RY	TD	BNS	ENB	TRP	CNR	CP
Symbols							
RY	0.046534	0.041665	0.041987	0.035117	0.033904	0.051713	0.036362
TD	0.041665	0.046988	0.042946	0.035584	0.034840	0.051418	0.037505
BNS	0.041987	0.042946	0.050607	0.038373	0.036890	0.053670	0.038373
ENB	0.035117	0.035584	0.038373	0.067828	0.047496	0.049610	0.039009
TRP	0.033904	0.034840	0.036890	0.047496	0.057689	0.047372	0.035655
CNR	0.051713	0.051418	0.053670	0.049610	0.047372	0.262292	0.061414

```
CP      0.036362  0.037505  0.038373  0.039009  0.035655  0.061414  0.077505
TRI     0.026703  0.026470  0.027277  0.025568  0.024642  0.037627  0.029533
FNV     0.011844  0.010843  0.012406  0.014481  0.014739  0.019318  0.012318
BCE     0.025346  0.025679  0.027595  0.027467  0.027868  0.031723  0.025891
L       0.034499  0.035456  0.034749  0.032450  0.031956  0.062689  0.038765
QSR     0.038831  0.040126  0.044065  0.048632  0.042336  0.058290  0.043558
CSU     0.046644  0.048589  0.049589  0.049589  0.052014  0.111326  0.053324
AAPL    0.026219  0.027723  0.027805  0.028103  0.025755  0.047196  0.032303
WMT     0.011345  0.011444  0.011288  0.011258  0.010927  0.015803  0.014857
TSLA    0.031429  0.032213  0.033040  0.038388  0.031184  0.066814  0.039484
GE      0.036562  0.038864  0.038606  0.035408  0.033041  0.068557  0.040649
AMZN    0.023023  0.023640  0.024218  0.023468  0.022326  0.047111  0.031517

Symbols       TRI       FNV       BCE         L       QSR       CSU      AAPL  \
Symbols
RY      0.026703  0.011844  0.025346  0.034499  0.038831  0.046644  0.026219
TD      0.026470  0.010843  0.025679  0.035456  0.040126  0.048589  0.027723
BNS     0.027277  0.012406  0.027595  0.034749  0.044065  0.049589  0.027805
ENB     0.025568  0.014481  0.027467  0.032450  0.048632  0.049589  0.028103
TRP     0.024642  0.014739  0.027868  0.031956  0.042336  0.052014  0.025755
CNR     0.037627  0.019318  0.031723  0.062689  0.058290  0.111326  0.047196
CP      0.029533  0.012318  0.025891  0.038765  0.043558  0.053324  0.032303
TRI     0.040290  0.010836  0.020682  0.026831  0.032422  0.032901  0.022442
FNV     0.010836  0.118585  0.012342  0.009400  0.002969  0.021295  0.010047
BCE     0.020682  0.012342  0.031460  0.023733  0.029458  0.031290  0.019459
L       0.026831  0.009400  0.023733  0.057556  0.040730  0.063643  0.031329
QSR     0.032422  0.002969  0.029458  0.040730  0.106559  0.067324  0.036783
CSU     0.032901  0.021295  0.031290  0.063643  0.067324  0.366705  0.040693
AAPL    0.022442  0.010047  0.019459  0.031329  0.036783  0.040693  0.076588
WMT     0.011853  0.003090  0.010544  0.016967  0.014193  0.020880  0.015693
TSLA    0.028720  0.013028  0.021452  0.035642  0.057864  0.063799  0.043987
GE      0.027112  0.005856  0.024462  0.043041  0.047073  0.063046  0.033522
AMZN    0.021372  0.005416  0.016312  0.027899  0.028877  0.043279  0.037421

Symbols       WMT      TSLA        GE      AMZN
Symbols
RY      0.011345  0.031429  0.036562  0.023023
TD      0.011444  0.032213  0.038864  0.023640
BNS     0.011288  0.033040  0.038606  0.024218
ENB     0.011258  0.038388  0.035408  0.023468
TRP     0.010927  0.031184  0.033041  0.022326
CNR     0.015803  0.066814  0.068557  0.047111
CP      0.014857  0.039484  0.040649  0.031517
TRI     0.011853  0.028720  0.027112  0.021372
FNV     0.003090  0.013028  0.005856  0.005416
BCE     0.010544  0.021452  0.024462  0.016312
L       0.016967  0.035642  0.043041  0.027899
QSR     0.014193  0.057864  0.047073  0.028877
```

```
CSU      0.020880  0.063799  0.063046  0.043279
AAPL     0.015693  0.043987  0.033522  0.037421
WMT      0.035222  0.014016  0.014766  0.014567
TSLA     0.014016  0.296999  0.041491  0.052414
GE       0.014766  0.041491  0.095521  0.029331
AMZN     0.014567  0.052414  0.029331  0.098437
Expected portfolio return: 0.1033808683174139
Expected variance: 0.1974185825877417
```




```
Optimal weights: [0.    0.    0.    0.    0.    0.    0.    0.    0.18  0.    0.
0.
 0.    0.275 0.166 0.096 0.    0.284]
Expected return, volatility and Sharpe ratio: [0.23187663 0.19427662 1.19353851]
```


Expected Return
0.20
0.15
0.10
0.05
0.16
0.18
0.20
0.22
0.24
Expected Volatility
Sharpe Ratio
0.8
0.7
0.6
0.5
0.4
0.3
0.2

[ ]:

[ ]: