Final Paper

May 2, 2020

[1]:
```
%%html
<a href="https://en.wikipedia.org/wiki/Risk_parity">Risk Parity Wikipedia</a>
```

```
<IPython.core.display.HTML object>
```

[2]:
```
import pandas as pd
import pandas_datareader.data as web
import numpy as np
import html
import datetime
import math
import matplotlib.pyplot as plt
from scipy.optimize import minimize
import scipy.optimize as sco
from pandas.plotting import register_matplotlib_converters
TOLERANCE = 1e-10
```

[3]:
```
np.random.seed(0)
register_matplotlib_converters()
```

[4]:
```
# import stock adj close price from Yahoo Finance
# '^IXIC' = NASDAQ Composite
# ^DJI = Dow Jones (U.S.)
# ^GSPC = S&P 500 (U.S.)
# '^N225' = Nikkie 225 (JPN)
# ^HSI = Hang Seng (Hong Kong)
# ^GDAXI = DAX (Germany)
# TY=F   = 10 year U.S. Treasury Note Future
# ^TNX = Treasury Yield 10 Years
# GC=F = Gold Future
# SI=F = Silver Future
# 'CL=F' Crude Oil Jun 20 Future
# BTC=F Bitcoin Future
# BRK-A Berkshire Hathaway

yahoo_tickers = [
                'TY=F','^GSPC', 'AMZN','JNJ',
```

```
                'GC=F','^N225'
                ]

start_date = datetime.datetime(2019, 5, 1)

end_date = datetime.datetime(2020, 5, 1)

prices = pd.DataFrame([web.DataReader(t,'yahoo', start_date, end_date).loc[:,␣
↪'Adj Close']
                       for t in yahoo_tickers],index=yahoo_tickers).T.
↪asfreq('B').ffill()

prices
```

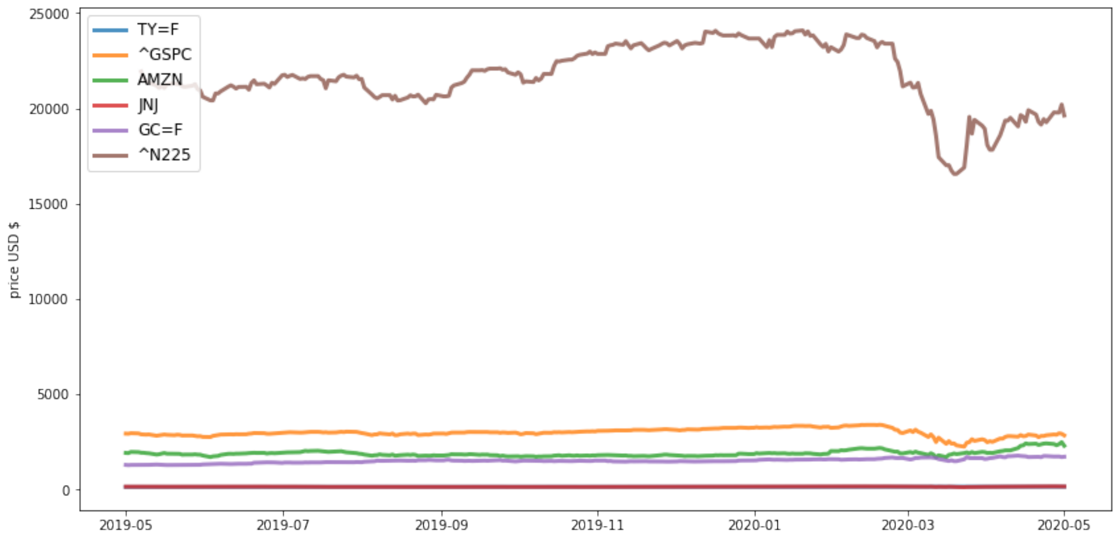
[4]:

```
                 TY=F        ^GSPC         AMZN          JNJ         GC=F  \
Date
2019-05-01  123.578003  2923.729980  1911.520020  138.093307  1281.400024
2019-05-02  123.171997  2917.520020  1900.819946  137.441513  1269.699951
2019-05-03  123.344002  2945.639893  1962.459961  138.151672  1279.199951
2019-05-06  123.625000  2932.469971  1950.550049  138.229492  1281.699951
2019-05-07  124.015999  2884.050049  1921.000000  136.167099  1283.500000
...                ...          ...          ...          ...          ...
2020-04-27  138.609375  2878.479980  2376.000000  154.289993  1720.300049
2020-04-28  139.015625  2863.389893  2314.080078  151.389999  1721.000000
2020-04-29  139.250000  2939.510010  2372.709961  150.240005  1722.199951
2020-04-30  139.000000  2912.429932  2474.000000  150.039993  1695.400024
2020-05-01  139.062500  2830.709961  2286.040039  148.289993  1710.199951

                   ^N225
Date
2019-05-01           NaN
2019-05-02           NaN
2019-05-03           NaN
2019-05-06           NaN
2019-05-07  21923.720703
...                  ...
2020-04-27  19783.220703
2020-04-28  19771.189453
2020-04-29  19771.189453
2020-04-30  20193.689453
2020-05-01  19619.349609

[263 rows x 6 columns]
```

[5]:

```
# prices from 05-01-2019 to 05-01-2020
plt.figure(figsize=(14, 7))
for c in prices.columns.values:
    plt.plot(prices.index, prices[c], lw=3, alpha=0.8,label=c)
```

```
plt.legend(loc='upper left', fontsize=12)
plt.ylabel('price USD $')
plt.show()
```



[6]:
```
# percentage change of returns

returns = prices.pct_change()
plt.figure(figsize=(14, 7))
for c in returns.columns.values:
    plt.plot(returns.index, returns[c], lw=3, alpha=0.8,label=c)
plt.legend(loc='upper right', fontsize=12)
plt.ylabel('daily returns')
plt.show()
```

0.1 Risk Free Rate

https://www.treasury.gov/resource-center/data-chart-center/interest-rates/pages/textview.aspx?data=yield

[7]:
```
returns = prices.pct_change()
mean_returns = returns.mean()
cov_matrix = returns.cov()
num_portfolios = 50000
risk_free_rate = 0.17 / 100
```

[8]:
```
def portfolio_annualised_performance(weights, mean_returns, cov_matrix):
    returns = np.sum(mean_returns*weights ) *252
    std = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights))) * np.sqrt(252)
    return std, returns

def random_portfolios(num_portfolios, mean_returns, cov_matrix, risk_free_rate):
    results = np.zeros((3,num_portfolios))
    weights_record = []
    for i in range(num_portfolios):
        weights = np.random.random(len(yahoo_tickers))
        weights /= np.sum(weights)
        weights_record.append(weights)
        portfolio_std_dev, portfolio_return =␣
 ↪portfolio_annualised_performance(weights, mean_returns, cov_matrix)
        results[0,i] = portfolio_std_dev
        results[1,i] = portfolio_return
        results[2,i] = (portfolio_return - risk_free_rate) / portfolio_std_dev
    return results, weights_record
```

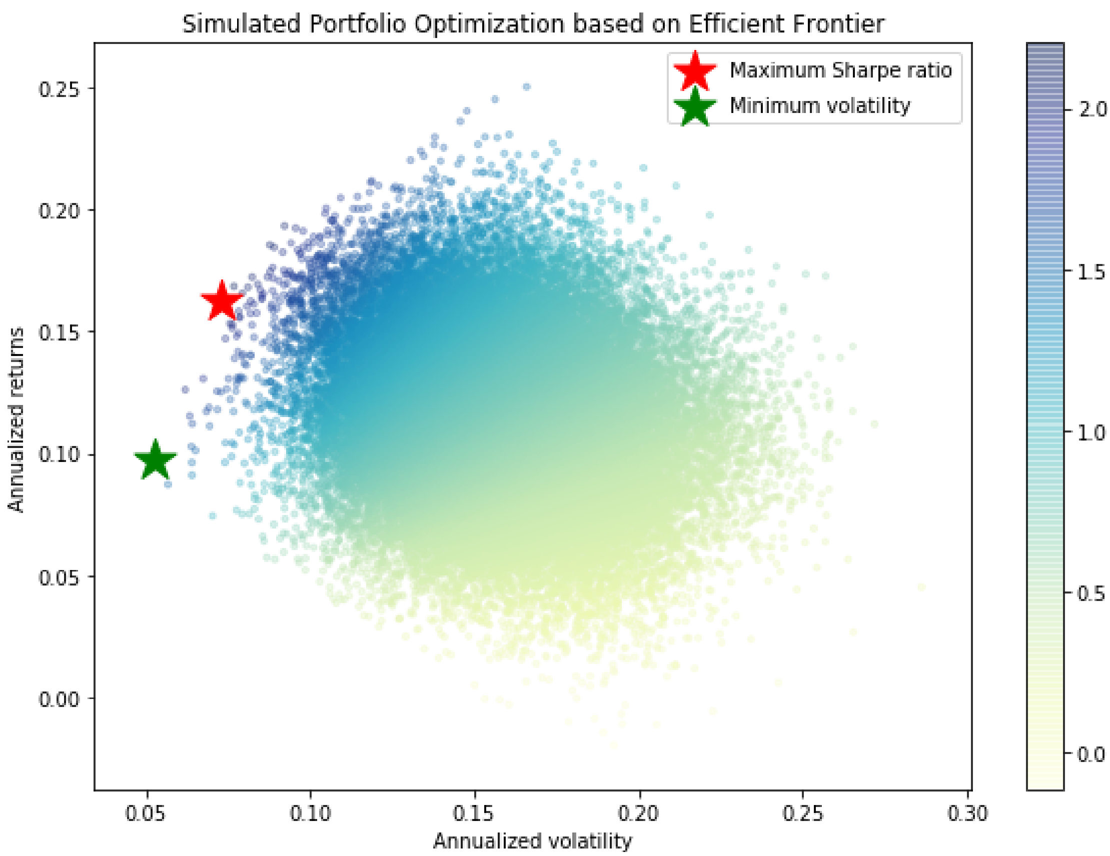
[9]:
```
def display_simulated_ef_with_random(mean_returns, cov_matrix, num_portfolios,␣
 ↪risk_free_rate):
    results, weights = random_portfolios(num_portfolios,mean_returns,␣
 ↪cov_matrix, risk_free_rate)

    max_sharpe_idx = np.argmax(results[2])
    sdp, rp = results[0,max_sharpe_idx], results[1,max_sharpe_idx]
    max_sharpe_allocation = pd.DataFrame(weights[max_sharpe_idx],index=prices.
 ↪columns,columns=['allocation'])
    max_sharpe_allocation.allocation = [round(i*100,2)for i in␣
 ↪max_sharpe_allocation.allocation]
    max_sharpe_allocation = max_sharpe_allocation.T

    min_vol_idx = np.argmin(results[0])
    sdp_min, rp_min = results[0,min_vol_idx], results[1,min_vol_idx]
```

```
    min_vol_allocation = pd.DataFrame(weights[min_vol_idx],index=prices.
 ↪columns,columns=['allocation'])
    min_vol_allocation.allocation = [round(i*100,2)for i in min_vol_allocation.
 ↪allocation]
    min_vol_allocation = min_vol_allocation.T

    print ("-"*80)
    print ("Maximum Sharpe Ratio Portfolio Allocation\n")
    print ("Annualized Return:", round(rp,2))
    print ("Annualized Volatility:", round(sdp,2))
    print ("\n")
    print (max_sharpe_allocation)
    print ("-"*80)
    print ("Minimum Volatility Portfolio Allocation\n")
    print ("Annualized Return:", round(rp_min,2))
    print ("Annualized Volatility:", round(sdp_min,2))
    print ("\n")
    print (min_vol_allocation)

    plt.figure(figsize=(10, 7))
    plt.scatter(results[0,:],results[1,:],c=results[2,:],cmap='YlGnBu',␣
 ↪marker='o', s=10, alpha=0.3)
    plt.colorbar()
    plt.scatter(sdp,rp,marker='*',color='r',s=500, label='Maximum Sharpe ratio')
    plt.scatter(sdp_min,rp_min,marker='*',color='g',s=500, label='Minimum␣
 ↪volatility')
    plt.title('Simulated Portfolio Optimization based on Efficient Frontier')
    plt.xlabel('Annualized volatility')
    plt.ylabel('Annualized returns')
    plt.legend(labelspacing=0.8)
```

[10]:
```
display_simulated_ef_with_random(mean_returns, cov_matrix, num_portfolios,␣
 ↪risk_free_rate)
```

```
--------------------------------------------------------------------------------
Maximum Sharpe Ratio Portfolio Allocation

Annualized Return: 0.16
Annualized Volatility: 0.07


             TY=F  ^GSPC   AMZN   JNJ  GC=F  ^N225
allocation  53.77   1.53  12.64  8.85  21.5   1.71
--------------------------------------------------------------------------------
Minimum Volatility Portfolio Allocation

Annualized Return: 0.1
Annualized Volatility: 0.05
```

```
            TY=F  ^GSPC  AMZN   JNJ  GC=F  ^N225
allocation  71.07  11.82  0.23  0.31  6.71   9.86
```



[11]:
```
# constraints of weights to sum up to 100%
constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
```

[12]:
```
def neg_sharpe_ratio(weights, mean_returns, cov_matrix, risk_free_rate):
    p_var, p_ret = portfolio_annualised_performance(weights, mean_returns,␣
 ↪cov_matrix)
    return -(p_ret - risk_free_rate) / p_var

def max_sharpe_ratio(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})
    bound = (0.0,1.0)
    bounds = tuple(bound for asset in range(num_assets))
    result = sco.minimize(neg_sharpe_ratio, num_assets*[1./num_assets,],␣
 ↪args=args,
```

```
                                method='SLSQP', bounds=bounds, constraints=constraints)
    return result
```

[13]:
```
def portfolio_volatility(weights, mean_returns, cov_matrix):
    return portfolio_annualised_performance(weights, mean_returns,␣
 ↪cov_matrix)[0]

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

    result = sco.minimize(portfolio_volatility, num_assets*[1./num_assets,],␣
 ↪args=args,
                                method='SLSQP', bounds=bounds, constraints=constraints)

    return result
```

[14]:
```
def efficient_return(mean_returns, cov_matrix, target):
    num_assets = len(mean_returns)
    args = (mean_returns, cov_matrix)

    def portfolio_return(weights):
        return portfolio_annualised_performance(weights, mean_returns,␣
 ↪cov_matrix)[1]

    constraints = ({'type': 'eq', 'fun': lambda x: portfolio_return(x) -␣
 ↪target},
                   {'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
    bounds = tuple((0,1) for asset in range(num_assets))
    result = sco.minimize(portfolio_volatility, num_assets*[1./num_assets,],␣
 ↪args=args, method='SLSQP', bounds=bounds, constraints=constraints)
    return result


def efficient_frontier(mean_returns, cov_matrix, returns_range):
    efficients = []
    for ret in returns_range:
        efficients.append(efficient_return(mean_returns, cov_matrix, ret))
    return efficients
```

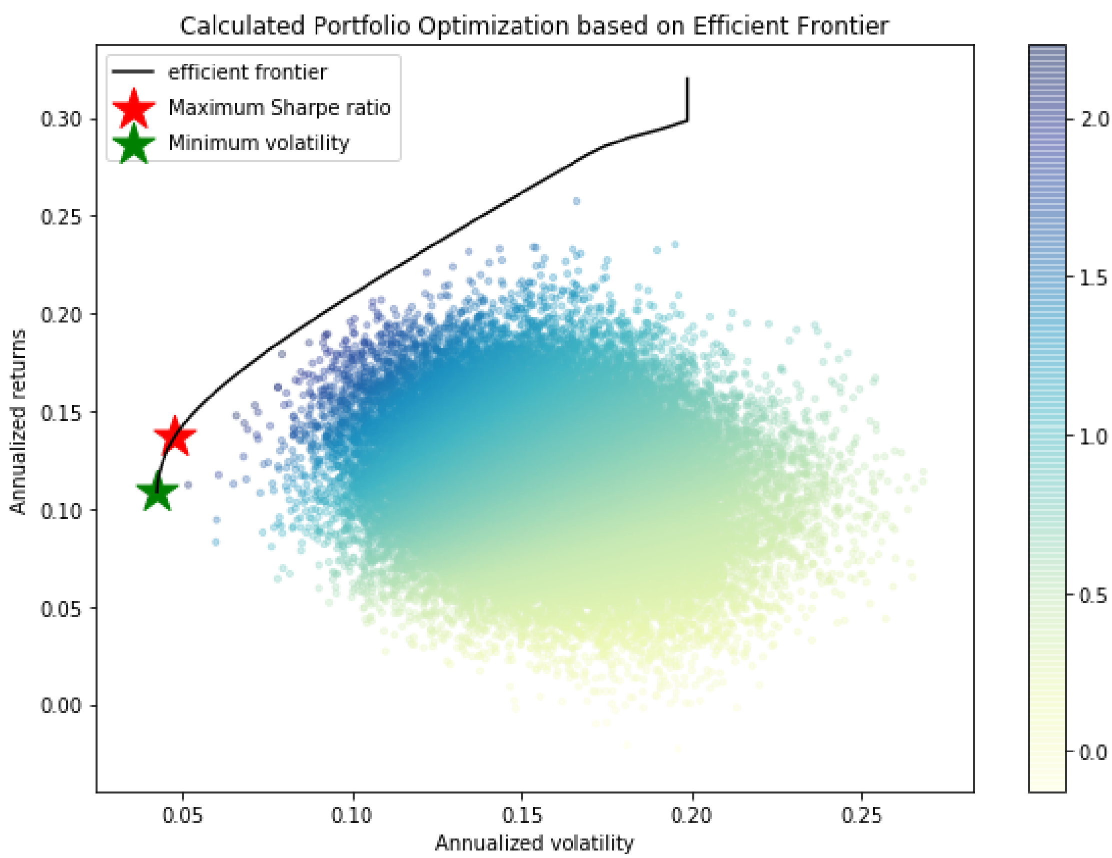
[15]:
```
def display_calculated_ef_with_random(mean_returns, cov_matrix, num_portfolios,␣
 ↪risk_free_rate):
    results, _ = random_portfolios(num_portfolios,mean_returns, cov_matrix,␣
 ↪risk_free_rate)
```

```
    max_sharpe = max_sharpe_ratio(mean_returns, cov_matrix, risk_free_rate)
    sdp, rp = portfolio_annualised_performance(max_sharpe['x'], mean_returns,␣
↪cov_matrix)
    max_sharpe_allocation = pd.DataFrame(max_sharpe.x,index=prices.
↪columns,columns=['allocation'])
    max_sharpe_allocation.allocation = [round(i*100,2)for i in␣
↪max_sharpe_allocation.allocation]
    max_sharpe_allocation = max_sharpe_allocation.T

    min_vol = min_variance(mean_returns, cov_matrix)
    sdp_min, rp_min = portfolio_annualised_performance(min_vol['x'],␣
↪mean_returns, cov_matrix)
    min_vol_allocation = pd.DataFrame(min_vol.x,index=prices.
↪columns,columns=['allocation'])
    min_vol_allocation.allocation = [round(i*100,2)for i in min_vol_allocation.
↪allocation]
    min_vol_allocation = min_vol_allocation.T

    print ("-"*80)
    print ("Maximum Sharpe Ratio Portfolio Allocation\n")
    print ("Annualised Return:", round(rp,2))
    print ("Annualised Volatility:", round(sdp,2))
    print ("\n")
    print (max_sharpe_allocation)
    print ("-"*80)
    print ("Minimum Volatility Portfolio Allocation\n")
    print ("Annualised Return:", round(rp_min,2))
    print ("Annualised Volatility:", round(sdp_min,2))
    print ("\n")
    print (min_vol_allocation)

    plt.figure(figsize=(10, 7))
    plt.scatter(results[0,:],results[1,:],c=results[2,:],cmap='YlGnBu',␣
↪marker='o', s=10, alpha=0.3)
    plt.colorbar()
    plt.scatter(sdp,rp,marker='*',color='r',s=500, label='Maximum Sharpe ratio')
    plt.scatter(sdp_min,rp_min,marker='*',color='g',s=500, label='Minimum␣
↪volatility')

    target = np.linspace(rp_min, 0.32, 50)
    efficient_portfolios = efficient_frontier(mean_returns, cov_matrix, target)
    plt.plot([p['fun'] for p in efficient_portfolios], target, linestyle='-',␣
↪color='black', label='efficient frontier')
    plt.title('Calculated Portfolio Optimization based on Efficient Frontier')
    plt.xlabel('Annualized volatility')
    plt.ylabel('Annualized returns')
```

```
    plt.legend(labelspacing=0.8)
```

[16]:
```
display_calculated_ef_with_random(mean_returns, cov_matrix, num_portfolios,␣
 ↪risk_free_rate)
```

```
--------------------------------------------------------------------------------
Maximum Sharpe Ratio Portfolio Allocation

Annualised Return: 0.14
Annualised Volatility: 0.05


             TY=F  ^GSPC  AMZN   JNJ  GC=F  ^N225
allocation  79.91    0.0  8.62  4.63  6.84    0.0
--------------------------------------------------------------------------------
Minimum Volatility Portfolio Allocation

Annualised Return: 0.11
Annualised Volatility: 0.04


             TY=F  ^GSPC  AMZN   JNJ  GC=F  ^N225
allocation  85.42   6.12  3.48  2.74   0.0   2.25
```



[17]:
```
def display_ef_with_selected(mean_returns, cov_matrix, risk_free_rate):
    max_sharpe = max_sharpe_ratio(mean_returns, cov_matrix, risk_free_rate)
    sdp, rp = portfolio_annualised_performance(max_sharpe['x'], mean_returns,␣
↪cov_matrix)
    max_sharpe_allocation = pd.DataFrame(max_sharpe.x,index=prices.
↪columns,columns=['allocation'])
    max_sharpe_allocation.allocation = [round(i*100,2)for i in␣
↪max_sharpe_allocation.allocation]
    max_sharpe_allocation = max_sharpe_allocation.T

    min_vol = min_variance(mean_returns, cov_matrix)
    sdp_min, rp_min = portfolio_annualised_performance(min_vol['x'],␣
↪mean_returns, cov_matrix)
    min_vol_allocation = pd.DataFrame(min_vol.x,index=prices.
↪columns,columns=['allocation'])
    min_vol_allocation.allocation = [round(i*100,2)for i in min_vol_allocation.
↪allocation]
    min_vol_allocation = min_vol_allocation.T

    an_vol = np.std(returns) * np.sqrt(252)
    an_rt = mean_returns * 252

    print ("-"*80)
    print ("Maximum Sharpe Ratio Portfolio Allocation\n")
    print ("Annualized Return:", round(rp,2))
    print ("Annualized Volatility:", round(sdp,2))
    print ("\n")
    print (max_sharpe_allocation)
    print ("-"*80)
    print ("Minimum Volatility Portfolio Allocation\n")
    print ("Annualized Return:", round(rp_min,2))
    print ("Annualized Volatility:", round(sdp_min,2))
    print ("\n")
    print (min_vol_allocation)
    print ("-"*80)
    print ("Individual Stock Returns and Volatility\n")
    for i, txt in enumerate(prices.columns):
        print (txt,":","Annualized return",round(an_rt[i],2),", Annualized␣
↪volatility:",round(an_vol[i],2))
    print ("-"*80)

    fig, ax = plt.subplots(figsize=(10, 7))
    ax.scatter(an_vol,an_rt,marker='o',s=200)

    for i, txt in enumerate(prices.columns):
```

```
        ax.annotate(txt, (an_vol[i],an_rt[i]), xytext=(10,0),␣
↪textcoords='offset points')
    ax.scatter(sdp,rp,marker='*',color='r',s=500, label='Maximum Sharpe ratio')
    ax.scatter(sdp_min,rp_min,marker='*',color='g',s=500, label='Minimum␣
↪volatility')

    target = np.linspace(rp_min, 0.34, 50)
    efficient_portfolios = efficient_frontier(mean_returns, cov_matrix, target)
    ax.plot([p['fun'] for p in efficient_portfolios], target, linestyle='-',␣
↪color='black', label='efficient frontier')
    ax.set_title('Portfolio Optimization with Individual Stocks')
    ax.set_xlabel('Annualized volatility')
    ax.set_ylabel('Annualized returns')
    ax.legend(labelspacing=0.8)
```

[18]:
```
display_ef_with_selected(mean_returns, cov_matrix, risk_free_rate)
```

```
--------------------------------------------------------------------------------
Maximum Sharpe Ratio Portfolio Allocation

Annualized Return: 0.14
Annualized Volatility: 0.05


             TY=F  ^GSPC  AMZN   JNJ  GC=F  ^N225
allocation  79.91    0.0  8.62  4.63  6.84    0.0
--------------------------------------------------------------------------------
Minimum Volatility Portfolio Allocation

Annualized Return: 0.11
Annualized Volatility: 0.04


             TY=F  ^GSPC  AMZN   JNJ  GC=F  ^N225
allocation  85.42   6.12  3.48  2.74   0.0   2.25
--------------------------------------------------------------------------------
Individual Stock Returns and Volatility

TY=F : Annualized return 0.12 , Annualized volatility: 0.06
^GSPC : Annualized return 0.02 , Annualized volatility: 0.32
AMZN : Annualized return 0.22 , Annualized volatility: 0.31
JNJ : Annualized return 0.11 , Annualized volatility: 0.3
GC=F : Annualized return 0.3 , Annualized volatility: 0.2
^N225 : Annualized return -0.08 , Annualized volatility: 0.23
--------------------------------------------------------------------------------
```