1) Let the market return be the target return. Estimate the expected deviation from market return, for the ten industry portfolios:

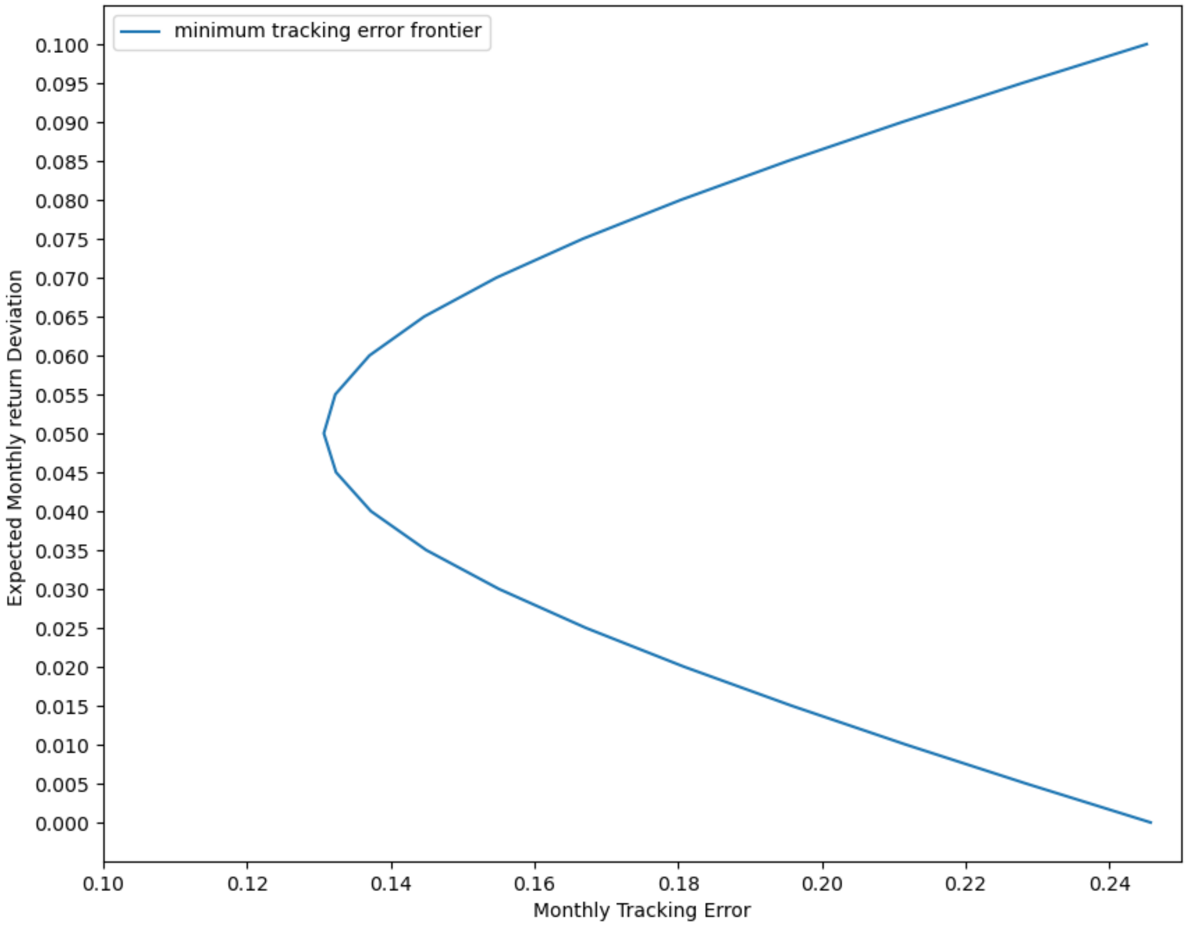
	industry	expected_deviation
0	NoDur	0.15475
1	Durbl	-0.01475
2	Manuf	0.26475
3	Enrgy	0.483083
4	HiTec	0.0181667
5	Telcm	0.133333
6	Shops	0.16825
7	Hlth	0.03575
8	Utils	0.159083
9	Other	-0.259

2) Also estimate the covariance matrix of return deviations, for the ten industry portfolios:

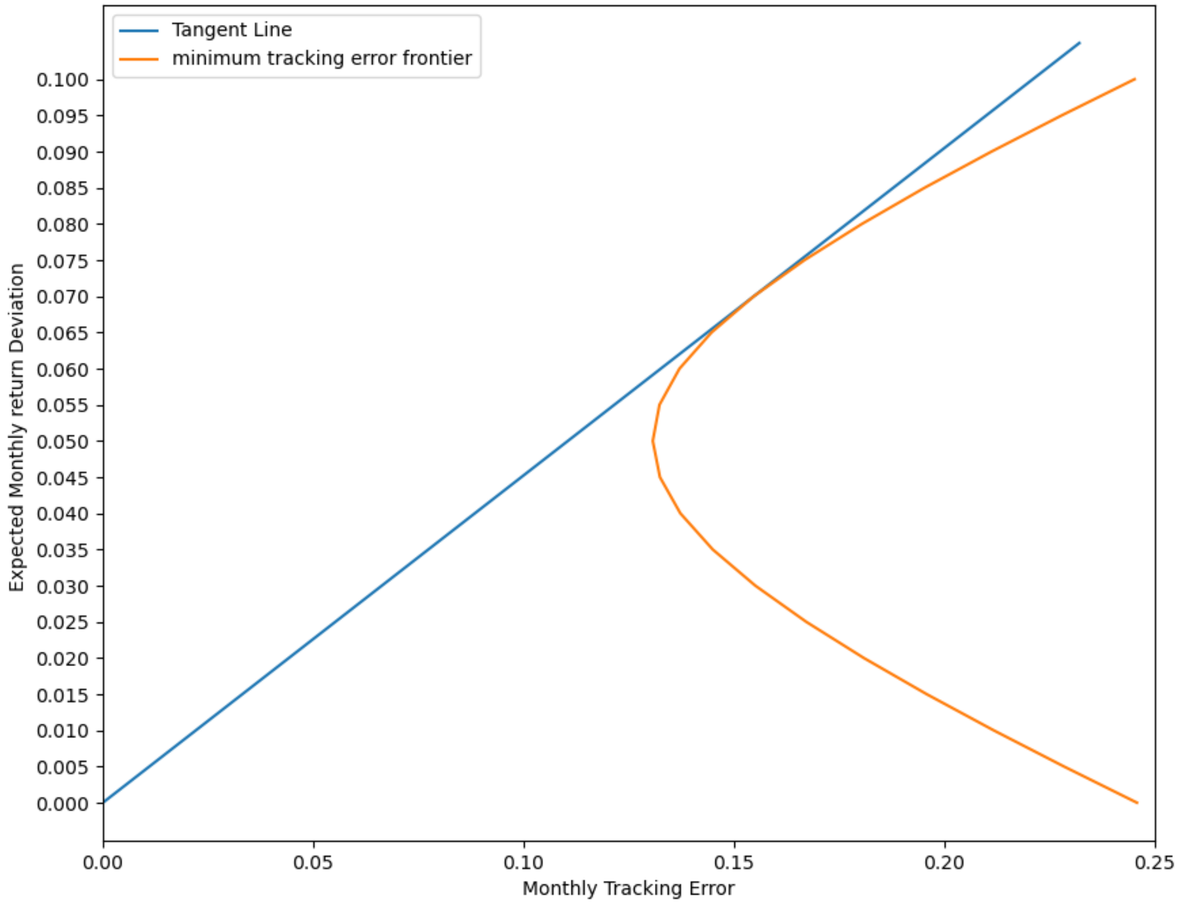
covariance:

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
NoDur	5.439696	-6.073035	-1.396192	-1.200533	-1.883151	1.538885	1.140741	3.815137	4.272002	-1.768738
Durbl	-6.073035	26.628901	4.908024	-3.481055	1.891577	-1.707625	-0.354335	-8.082946	-9.617490	4.385865
Manuf	-1.396192	4.908024	2.950499	1.666133	0.065267	-0.626416	-1.154597	-2.288900	-1.901412	0.358904
Enrgy	-1.200533	-3.481055	1.666133	19.274911	-1.516972	-1.040525	-3.710439	-2.485796	4.454368	-3.864826
HiTec	-1.883151	1.891577	0.065267	-1.516972	5.098746	-0.773294	-0.245350	-1.936284	-2.342839	-1.404050
Telcm	1.538885	-1.707625	-0.626416	-1.040525	-0.773294	4.682567	0.463797	0.693157	2.721477	-1.271778
Shops	1.140741	-0.354335	-1.154597	-3.710439	-0.245350	0.463797	4.452628	0.764510	-0.176666	-0.256987
Hlth	3.815137	-8.082946	-2.288900	-2.485796	-1.936284	0.693157	0.764510	7.820446	3.496136	-1.726842
Utils	4.272002	-9.617490	-1.901412	4.454368	-2.342839	2.721477	-0.176666	3.496136	12.267476	-4.055112
Other	-1.768738	4.385865	0.358904	-3.864826	-1.404050	-1.271778	-0.256987	-1.726842	-4.055112	4.503204

3) Plot the minimum-tracking-error frontier generated by the ten industry portfolios, with expected (monthly) return deviation on the vertical axis and (monthly) tracking error on the horizontal axis. This plot should cover the range from 0% to 0.1% on the vertical axis, in increments of 0.005% (or less).



4) Also plot the line starting from the origin that is tangent to the upper half of the minimum-tracking-error frontier, and calculate the information ratio and portfolio weights for the "tangency" portfolio.



5) Information ratio:

```
information ratio: [[0.45248754]]
```

6) portfolio weights for the "tangency" portfolio:

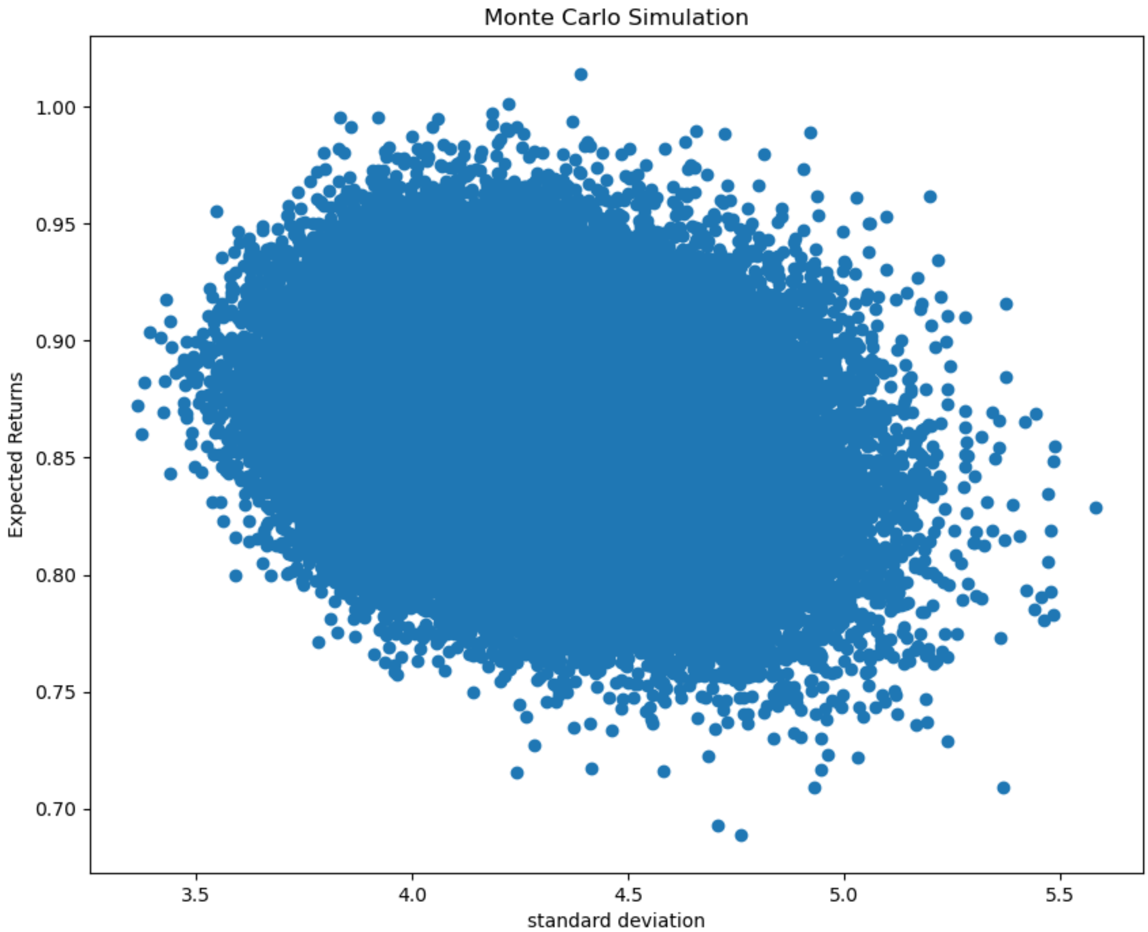
```
weight of tangency portfolio

         Weight
NoDur  0.052634
Durbl  0.000153
Manuf  0.137627
Enrgy  0.087032
HiTec  0.179353
Telcm  0.071074
Shops  0.106884
Hlth   0.102776
Utils  0.040162
Other  0.222304
```

Use the monthly returns of the ten industry portfolios to generate the minimum-variance frontier without short sales, using Monte Carlo simulation. Portfolio weights will be limited to the range [0, 1].

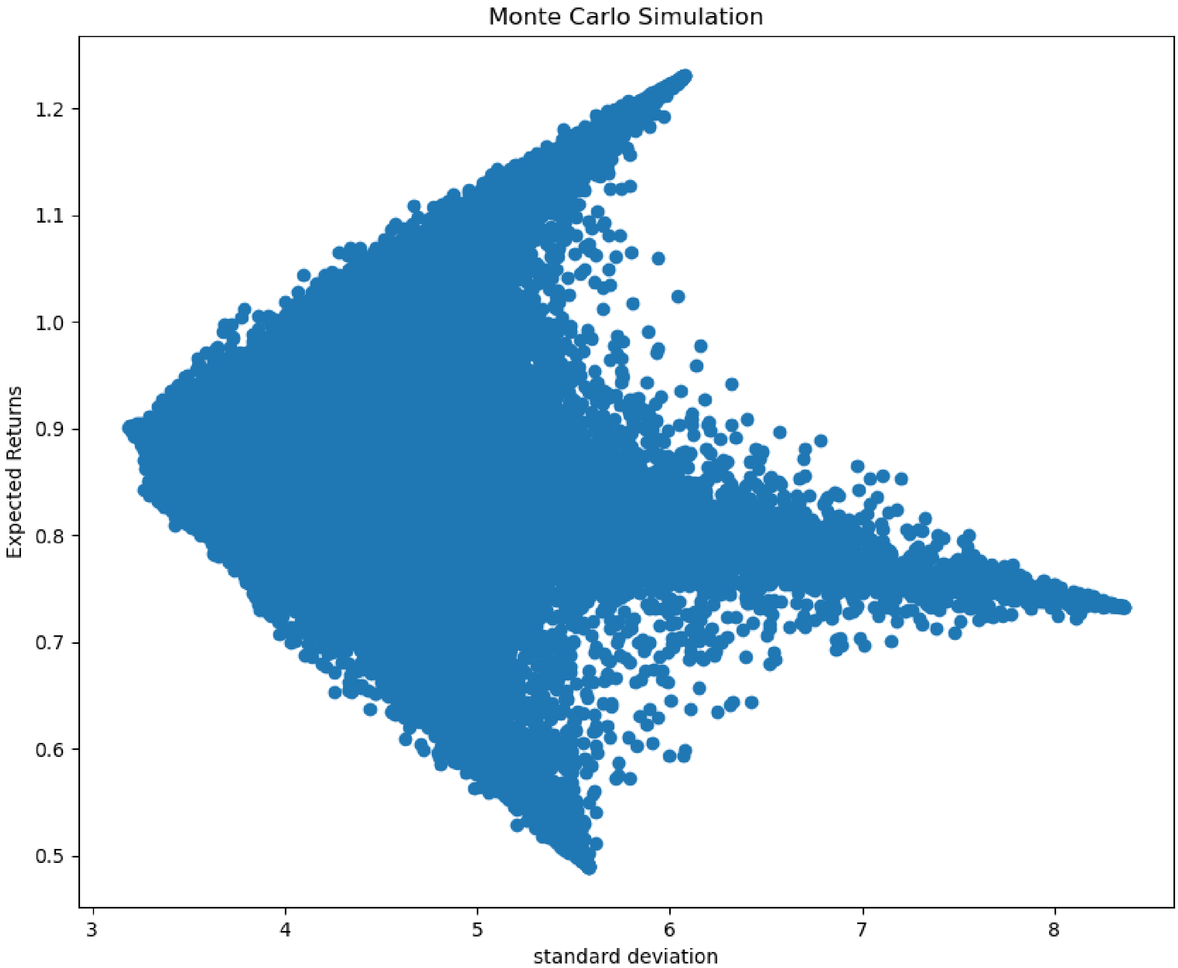
Randomly draw each element of **w**, the vector of portfolio weights, from the (standard) uniform distribution in the range [0, 1]. Divide **w** by the sum of the portfolio weights, to ensure that the portfolio weights sum to one. Use the normalised **w** to calculate the mean return and standard deviation of return for the simulated portfolio. Repeat this process until you have (at least) 10^5 data points.

7) Plot the data points with mean return on the vertical axis and standard deviation of return on the horizontal axis.



Repeat this entire process by simulating 1/w using the standard uniform distribution ⇒ take the reciprocal of the random draw from the standard uniform distribution as the portfolio weight.

8) Plot the new data points with mean return on the vertical axis and standard deviation of return on the horizontal axis.


Monte Carlo Simulation
Expected Returns
standard deviation
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
3
4
5
6
7
8

Appendix:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 1 14:51:26 2022
@author: XuebinLi
"""
import warnings
warnings.simplefilter("ignore", UserWarning)
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from tabulate import tabulate
from numpy.linalg import inv
import math
# create dataframe
#df_monthly_returns =
pd.read_excel('C:\\Users\\lixue\\OneDrive\\Desktop\\smu\\MQF\\Asset
Pricing\\lesson5\\Industry_Portfolios.xlsx')
#df_market =
pd.read_excel('C:\\Users\\lixue\\OneDrive\\Desktop\\smu\\MQF\\Asset
Pricing\\lesson5\\Market_Portfolio.xlsx')
df_monthly_returns = pd.read_excel('C:\\Users\\XuebinLi\\OneDrive - Linden
Shore
LLC\\Desktop\\smu\\New
folder\\Asset_Pricing_SMU\\Industry_Portfolios.xlsx')
df_market = pd.read_excel('C:\\Users\\XuebinLi\\OneDrive - Linden Shore
LLC\\Desktop\\smu\\New
folder\\Asset_Pricing_SMU\\Market_Portfolio.xlsx')
df = df_monthly_returns.sub(df_market['Market'],axis=0)
d = []
d_without_date = []
for p in df:
if "Date" not in p and "date" not in p:
d.append((p, df[p].mean()))
df_table_mean_std = pd.DataFrame(d, columns=('industry',
'expected_deviation'))
#covariance
df_cov = df.iloc[: , 1:]
df_cov = df_cov.cov()
np_df_cov = df_cov.to_numpy()
#vector mean
vector_mean = df_table_mean_std[["expected_deviation"]].to_numpy()
# transpose of returns
vector_mean_transpose = np.transpose(vector_mean)
#inverse covariance
df_cov_inverse = inv(df_cov)
#e
#weight = [1, 1, 1, 1, 1, 1, 1, 1, 1,1]
weight = np.ones(10)
#e transpose
weight_transpose = np.transpose(weight)
#alpha
alpha = np.matmul(vector_mean_transpose, df_cov_inverse)
alpha = np.matmul(alpha, weight)
#zetha
zelta = np.matmul(vector_mean_transpose, df_cov_inverse)
zelta = np.matmul(zelta, vector_mean)
#delta
delta = np.matmul(weight_transpose, df_cov_inverse)
```

```
delta = np.matmul(delta, weight)
#rmv mid line
rmv = alpha/delta
#riskfree_Rate
rf_rate = 0.0
#tangency portfolio
Rtg = (alpha * rf_rate - zelta)/(delta*rf_rate - alpha)
Rtg = Rtg[0][0]
Stg = -((zelta-
2*alpha*rf_rate+delta*rf_rate*rf_rate)**0.5)/(delta*(rf_rate-rmv))
#sharpe ratio
def sharpe_ratio(Rtg,rf_rate,Stg):
sharpe_ratio = (Rtg - rf_rate)/Stg
return sharpe_ratio
#weight of optimal portfolio
def weight_portfolio():
lmda = (Rtg - rf_rate)/(zelta - 2*alpha*rf_rate+delta*rf_rate*rf_rate)
lmda = lmda[0][0]
weight_Rtg1 = np.dot(lmda,df_cov_inverse)
weight_Rtg1 = lmda * df_cov_inverse
weight_Rtg2 = np.dot(rf_rate,weight)
weight_Rtg2 = np.reshape(weight_Rtg2, (10, 1))
weight_Rtg3 = np.subtract(vector_mean,weight_Rtg2)
weight_final = np.dot(weight_Rtg1,weight_Rtg3)
weight_pd = pd.DataFrame(data=weight_final, index =
df.columns[1:11],columns=['Weight'])
return weight_final, weight_pd
def
print_all(sharpe_ratio,weight_final,vector_mean,df_cov,df_table_mean_std):
print("information ratio:", sharpe_ratio, "\n" )
print("weight of tangency portfolio", "\n")
print(weight_final[1], "\n")
#question1: vector of mean and covariance
print("covariance:", "\n")
print(df_cov, "\n")
#question2: table with mean and std
print(tabulate(df_table_mean_std, headers = 'keys', tablefmt = 'psql'),
"\n")
def plot_all():
def my_range(start, end, step):
while start <= end:
yield start
start += step
# #risk-free line(PAGE 25 lecture)
yaxis2 = []
xaxis2 = []
for x2 in my_range(rf_rate, 0.11, 0.005):
stdplot2 = ((x2 - rf_rate)**2)/(zelta -
2*alpha*rf_rate+zelta*rf_rate*rf_rate)
stdplot2 = math.sqrt(stdplot2)
xaxis2 += [stdplot2]
yaxis2 += [x2]
plt.plot(xaxis2,yaxis2,label="Tangent Line")
plt.xlabel("Monthly Tracking Error")
plt.ylabel("Expected Monthly return Deviation ")
plt.yticks(np.arange(0, max(yaxis2), 0.005))
plt.legend()
yaxis = []
xaxis = []
for x in my_range(0, 0.105, 0.005):
```

```
stdplot = (1/delta) + (delta/(zelta*delta-(alpha*alpha))) * (x -
(alpha/delta))**2
stdplot = math.sqrt(stdplot)
xaxis += [stdplot]
yaxis += [x]
plt.plot(xaxis,yaxis,label="minimum tracking error frontier")
plt.xlabel("Monthly Tracking Error")
plt.ylabel("Expected Monthly return Deviation ")
plt.yticks(np.arange(0, max(yaxis), 0.005))
plt.xlim(0.0,0.25)
plt.legend()
#print_all(sharpe_ratio(Rtg,rf_rate,Stg),weight_portfolio(),vector_mean
,df_cov,df_table_mean_std)
#plot_all()
def monte_carlo_weight(sizee,industry):
#get weight 100000 * 10
#sum = 1 and value>0
matrix_weight = np.random.rand(industry,sizee)
matrix_weight = matrix_weight/matrix_weight.sum(axis=0)
matrix_weight_dd = np.divide(1,matrix_weight)
matrix_weight_dd = matrix_weight_dd/matrix_weight_dd.sum(axis=0)
return matrix_weight, matrix_weight_dd
#monte carlo simulatio
def monte_carlo(sizee,industry,matrix_weight):
# print(matrix_weight, "\n")
df_monthly_returns_np = df_monthly_returns.drop(['Date'], axis=1)
df_monthly_returns_numpy_mean = df_monthly_returns_np.mean().to_numpy()
returns_matrix = np.matmul(matrix_weight.T,df_monthly_returns_numpy_mean)
df_returns_matrix = pd.DataFrame(data = returns_matrix)
df_returns_matrix_mean = df_returns_matrix.sum(axis=1)
df_returns_matrix_mean = df_returns_matrix_mean.to_numpy()
df_cov_ri = df_monthly_returns_np.cov()
df_cov_ri = df_cov_ri.to_numpy()
wprimexv =
np.dot(matrix_weight.T[None,:],df_cov_ri).reshape(sizee,industry)
pt_var = (wprimexv *
matrix_weight.T[None,:].reshape(sizee,industry)).sum(axis=1)
pt_std = np.sqrt(pt_var)
plt.scatter(pt_std, returns_matrix)
plt.title('Monte Carlo Simulation')
plt.xlabel('standard deviation')
plt.ylabel('Expected Returns')
plt.show()
return True
monte_carlo(100000,10,monte_carlo_weight(100000,10)[0])

#monte_carlo(100000,10,monte_carlo_weight(100000,10)[1])
```