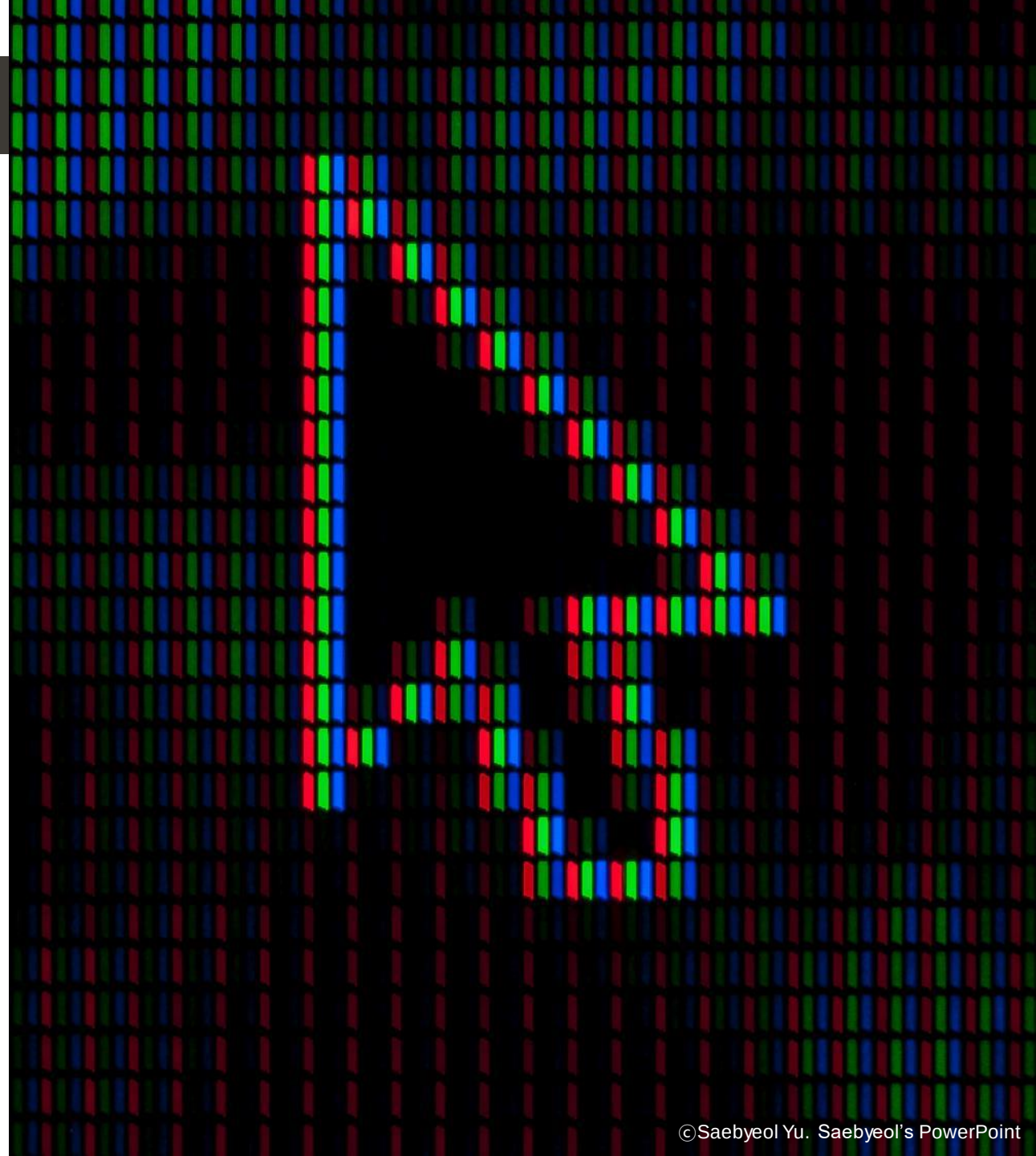
Financial Data Analysis

Simulating Asset Returns

2017100923
산업경영공학과
정솔

목차 a table of contents

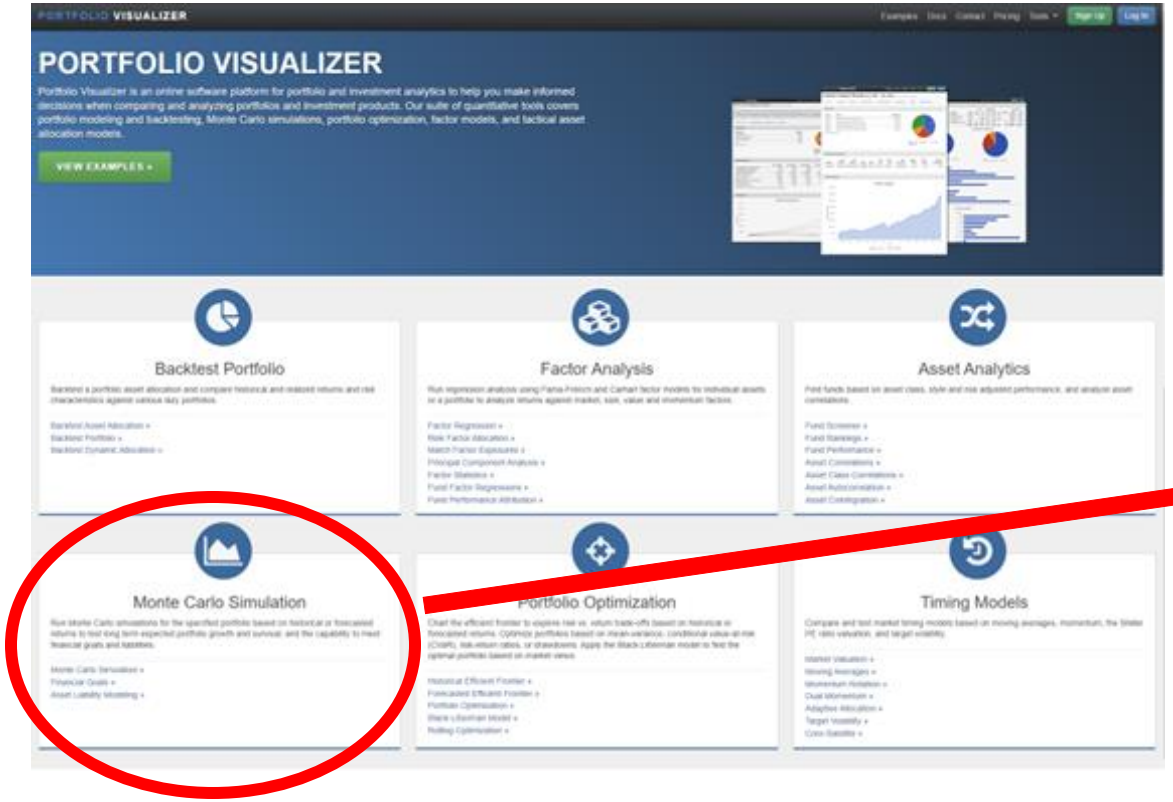


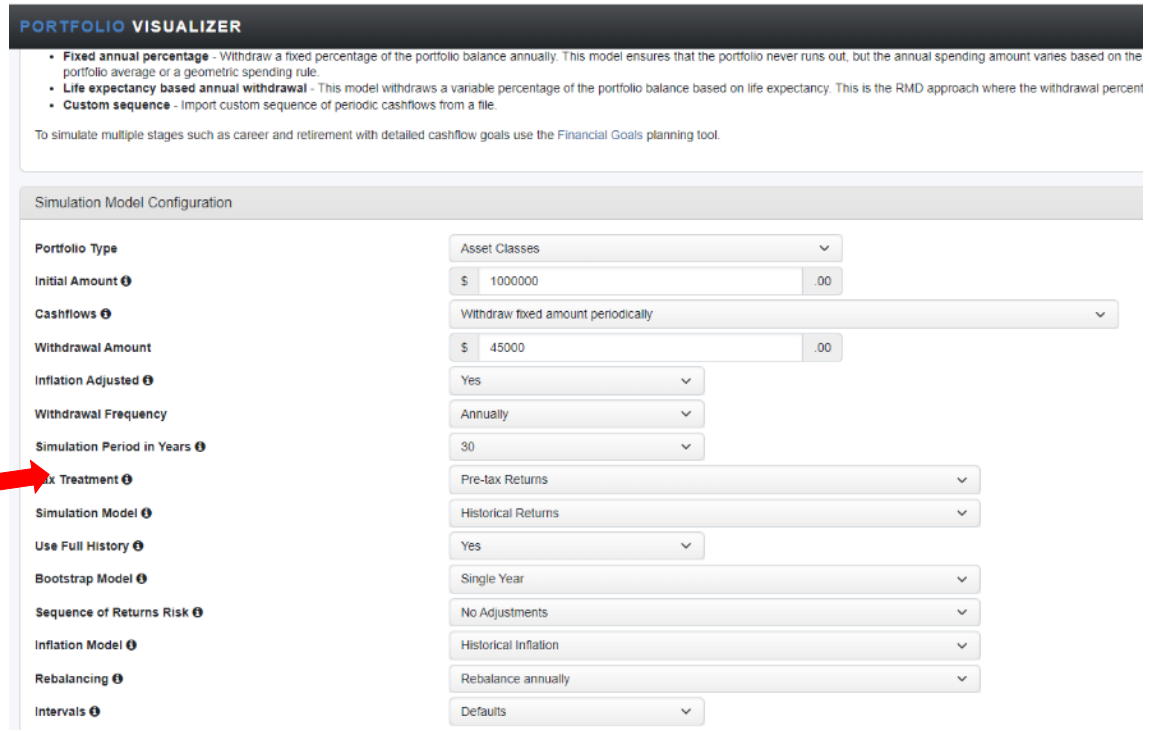
Part 1 Import Data

Part 1

Import Data

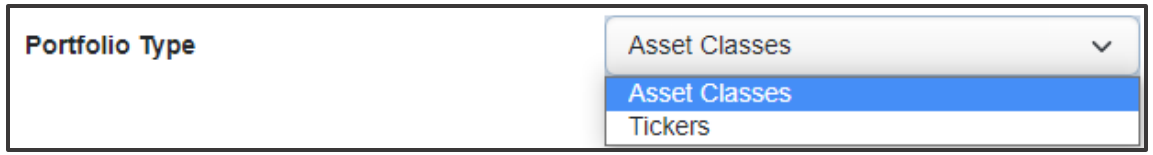
Reference





Portfolio Visualizer (https://www.portfoliovisualizer.com)

Part 1 Import Data



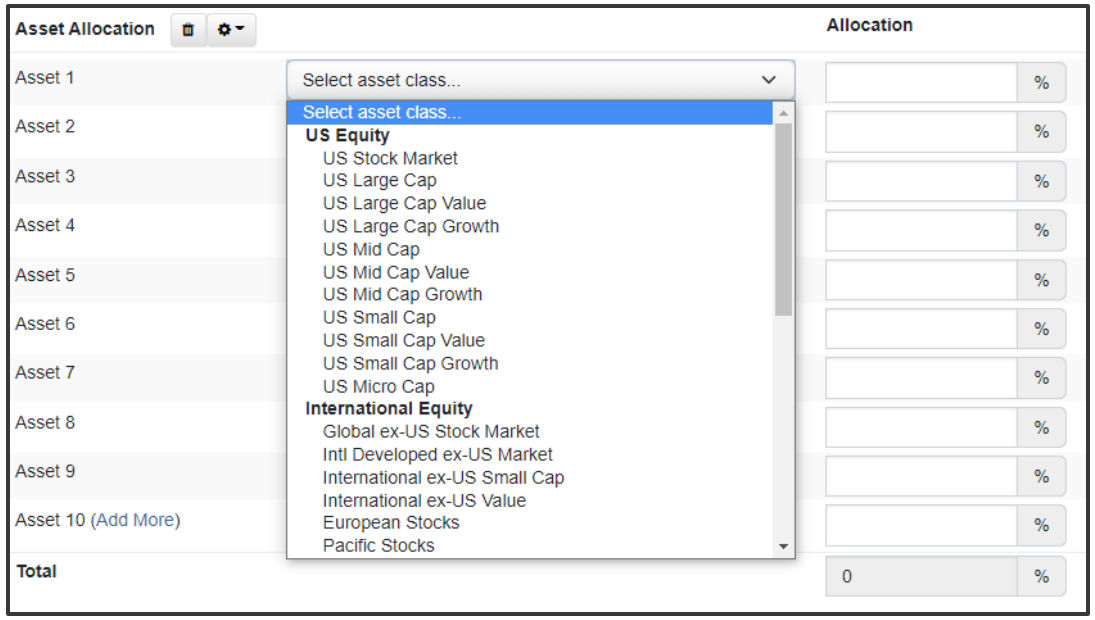

```
# select type
Portiolo_Type = input("\nPortiolo Type (Asset Classes or Tikers): ").lower()
while isinstance(Portiolo_Type, str):
  if Portiolo_Type in ['asset', 'classes', 'class', 'a', 'asset classes', 'asset class']:
    Portiolo_Type = 1
  elif Portiolo_Type in ['tiker', 'tikers', 't']:
    Portiolo_Type = 0
  else:
    print('\nenter "asset" or "tikers"')
    Portiolo_Type = input("Portiolo Type (Asset Classes or Tikers): ").lower()

# detail
print()
if Portiolo_Type == 1:
  print("Ex: ", *get_available_datasets()[:10])
  stocks = input("class: ")
else:
  print("Ex: CBA BHP TLS NAB WBC STO")
  stockList = input("stocks: ")
  except_list = [',', '/']

  while 1:
    if any(word in stockList for word in except_list):
      print("\nDo not include commas. Enter as in the above")
      stockList = input("stocks: ")
    else:
      break

  stocks = [stock + '.AX' for stock in list(stockList.split())]
```

- Portfolio의 종류 지정
- Visualizer에서는 'Asset Classes'와 'Tickers'로 구분
- 이와 모방하여 class와 ticker의 입력을 구분
- 입력의 편의를 위해 조건, 반복문 구축

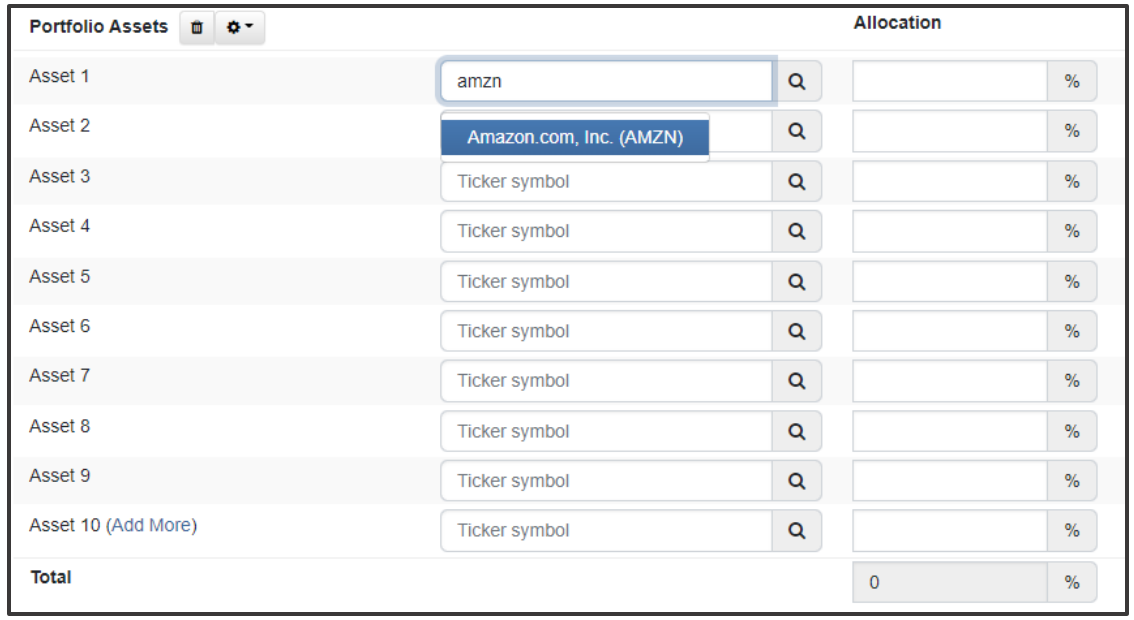
- 여기서는 모든 class를 다 불러올 수는 없어 Fama-French library의 산업군 데이터를 불러오고, 산업별로 portfolio 구성
- Class를 선택할 경우 "get_available_datasets"함수를 통해 입력 가능한 자료를 보여주고 입력을 받음

- Ticker를 선택할 경우 symbol을 입력 받아 list에 저장함
- Class에서 처럼 입력 조건에 맞지 않을 경우를 대비해 반복문 구축

Part 1 Import Data



Type을 Asset Classes로 지정했을 경우



Type을 Tickers로 지정했을 경우

Part 1 Import Data

```
if Portiolo_Type == 1: # asset classes
  returns, meanReturns, covMatrix, weights = get_data_c(stocks, startDate, endDate)

else: # tickers
  startDate_t = startDate - relativedelta(months=1)
  returns, meanReturns, covMatrix, weights = get_data_t(stocks, startDate_t, endDate)
```

```
# import data with class(fama-french)
def get_data_c(stocks, start, end):
  stockData = web.DataReader(stocks, 'famafrench', start, end)
  returns = stockData[0]/100
  meanReturns = returns.mean()
  covMatrix = returns.cov()

  n = returns.shape[1]
  weights = np.array([1/n for i in range(n)])

  return returns, meanReturns, covMatrix, weights


# import data with tikers
def get_data_t(stocks, start, end):
  stockData_0 = web.DataReader(stocks, 'yahoo', start, end)
  stockData = stockData_0['Adj Close']
  stockData_w = stockData.resample('M').agg('last')
  returns = stockData_w.pct_change()
  returns.drop(returns.index[[0]], inplace=True)
  meanReturns = returns.mean()
  covMatrix = returns.cov()

  n = len(meanReturns)
  weights = np.array([1/n for i in range(n)])

  return returns, meanReturns, covMatrix, weights
```

- 앞서 입력 받은 Portfolio type을 class의 경우 '1', ticker의 경우 '0'으로 구분

- 각각 함수를 다르게 정의하여 data import
(위) Asset class (famafrench):
get_data_c (stocks, start, end)

(아래) ticker:
get_data_t(stocks, start, end)

- Famafrench의 경우 퍼센트 데이터로 불러들여지기 때문에 100을 나눠 return을 저장
- mean 함수를 통해 수익률의 평균을 저장
- cov함수를 통해 행렬의 공분산 저장

- 산업군의 개수 또는 주식의 개수를 n에 저장해 동일한 비중의 weights array를 생성

Ex) n = 5, weights = array([0.2, 0.2, 0.2, 0.2, 0.2])

Part 1 Import Data


```
# date
Use_Full_History = input("\nUse_Full_History (Yes or No): ").lower()
while isinstance(Use_Full_History, str):
  if Use_Full_History in ['yes', 'y', '1']:
    Use_Full_History = 1
  elif Use_Full_History in ['no', 'n', '0']:
    Use_Full_History = 0
  else:
    print('\nenter "yes" or "no"')
    Use_Full_History = input("Use_Full_History (Yes or No): ").lower()

startDate, endDate = time(Use_Full_History)
```

```
def time (answer):
  if answer == 1:
    #endDate = dt.datetime.now()
    endDate = parse('2022-10-31')
    startDate = endDate - relativedelta(years=20)
    startDate = f"{startDate.year}-01-15"

    return parse(startDate), endDate

  else:
    Start_Year = input("Start_Year: ")
    End_Year = input("End_Year: ")

    startDate = parse(Start_Year)
    startDate = f"{startDate.year}-01-07"
    endDate = parse(End_Year)
    endDate = f"{endDate.year}-12-31"

    return parse(startDate), parse(endDate)
```

- (Visualizer) 학습에 사용할 데이터 기간 설정 가능
- Yes의 경우 홈페이지 내 기본값으로,
- No를 선택할 경우 시작일과 종료일을 선택할 수 있음

- 마찬가지로 yes와 no로 입력을 구분
- Yes의 경우 현재 날짜로 부터 20년 간의 데이터를 불러오게 되고,
- No이 경우 사용자가 입력한 기간 동안의 데이터를 불러옴

- 하지만 yes의 경우 현재 날짜가 아닌 22.10.31.로 되어있는데 이는 fama French library에서 해당 날짜까지만 데이터를 제공하고 있기 때문

Part 1 Import Data

```
Portiolo_Type = 1   # select 'asset classes'
stocks = '10_Industry_Portfolios'
startDate = '2000-01-07'
endDate = '2020-12-31'

if Portiolo_Type == 1:
  returns, meanReturns, covMatrix, weights = get_data_c(stocks, startDate, endDate)
else:
  startDate_t = startDate - relativedelta(months=1)
  returns, meanReturns, covMatrix, weights = get_data_t(stocks, startDate_t, endDate)
```

- Portfolio_type, stocks, startDate, EndDate을 다음과 같이 지정하고 함수 실행
- 시뮬레이션 및 평가 분석은 월별로 진행하기 때문에 “일(日)”은 무시

- Startdatae을 1일로 설정할 경우 datareader로 불러오는 채널에 따라(yahoo or fama-french) 해당 월 이전의 데이터를 불러오는 경우가 있어 1주일 정도 지난 시점인 7일로 설정

Part 1

Import Data

10_Industry_Portfolios의 월별 수익률

```
1 df(returns)
```

Date	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
2000-01	-0.0478	-0.0086	-0.0867	0.0091	-0.0470	-0.0398	-0.1136	0.0754	0.0606	-0.0469
2000-02	-0.0619	-0.0800	-0.0394	-0.0566	0.1818	-0.0347	-0.0374	-0.0292	-0.0726	-0.0712
2000-03	0.0776	0.1052	0.0748	0.1209	0.0395	0.0752	0.1332	0.0030	0.0577	0.1408
2000-04	-0.0182	0.0929	0.0145	-0.0192	-0.1069	-0.0787	-0.0447	0.0528	0.0760	-0.0312
2000-05	0.0722	-0.1323	-0.0166	0.0954	-0.1084	-0.1043	-0.0277	0.0395	0.0390	0.0340
...	...	...	...	...	...	...	...	...	...	...
2020-08	0.0445	0.4019	0.0633	-0.0107	0.1050	0.0551	0.0816	0.0245	-0.0225	0.0721
2020-09	-0.0198	-0.0897	-0.0002	-0.1490	-0.0511	-0.0212	-0.0387	-0.0148	-0.0027	-0.0297
2020-10	-0.0256	-0.0329	-0.0076	-0.0453	-0.0182	-0.0385	-0.0257	-0.0442	0.0449	-0.0190
2020-11	0.1002	0.3385	0.1386	0.2846	0.1083	0.1443	0.0838	0.0952	0.0263	0.1576
2020-12	0.0500	0.1565	0.0260	0.0616	0.0496	0.0529	0.0149	0.0476	0.0063	0.0531

252 rows × 10 columns

수익률 간의 공분산

```
1 df(covMatrix)
```

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
NoDur	0.001304	0.001765	0.001352	0.001388	0.001147	0.001208	0.001128	0.000931	0.000861	0.001387
Durbl	0.001765	0.007226	0.003507	0.003113	0.003739	0.002882	0.002832	0.001689	0.001187	0.003621
Manuf	0.001352	0.003507	0.002566	0.002430	0.002462	0.001894	0.001815	0.001311	0.001037	0.002394
Enrgy	0.001388	0.003113	0.002430	0.004772	0.002007	0.001828	0.001567	0.001282	0.001533	0.002280
HiTec	0.001147	0.003739	0.002462	0.002007	0.004792	0.002639	0.002146	0.001436	0.000726	0.002446
Telcm	0.001208	0.002882	0.001894	0.001828	0.002639	0.002769	0.001680	0.001219	0.000824	0.002016
Shops	0.001128	0.002832	0.001815	0.001567	0.002146	0.001680	0.002056	0.001067	0.000695	0.001946
Hlth	0.000931	0.001689	0.001311	0.001282	0.001436	0.001219	0.001067	0.001666	0.000737	0.001398
Utils	0.000861	0.001187	0.001037	0.001533	0.000726	0.000824	0.000695	0.000737	0.001726	0.001018
Other	0.001387	0.003621	0.002394	0.002280	0.002446	0.002016	0.001946	0.001398	0.001018	0.002866

```
1 df(meanReturns, columns = ['mean return']).T
```

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
mean return	0.008518	0.010363	0.008673	0.006177	0.007625	0.003913	0.008859	0.008143	0.008364	0.006554

산업군 별 평균 수익률

```
1 weights
```

```
array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1])
```

동일 가중 포트폴리오를 위한 가중치 / sum = 1

Part 2 Portfolio Optimization

Part 2 Portfolio Optimization

Import data: “10_Industry_Portfolios”

Portfolio Optimization

```
[109] 1 asset_list = returns.columns
      2 asset_num = len(asset_list)
      3 asset_list, asset_num
```

```
(Index(['NoDur', 'Durbl', 'Manuf', 'Enrgy', 'HiTec', 'Telcm', 'Shops', 'Hlth ',
        'Utils', 'Other'],
       dtype='object'), 10)
```

```
[112] 1 mu = returns.mean() * 12 # annualized
      2 df(mu).T
```

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
0	0.102214	0.124357	0.104071	0.074129	0.091505	0.046962	0.10631	0.097714	0.100367	0.078643

```
[113] 1 sigma = returns.cov() * 12 # annualized
      2 sigma
```

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
NoDur	0.015647	0.021177	0.016220	0.016659	0.013768	0.014496	0.013540	0.011173	0.010333	0.016647
Durbl	0.021177	0.086706	0.042087	0.037358	0.044862	0.034589	0.033981	0.020272	0.014249	0.043447
Manuf	0.016220	0.042087	0.030790	0.029156	0.029548	0.022723	0.021782	0.015728	0.012443	0.028728
Enrgy	0.016659	0.037358	0.029156	0.057266	0.024083	0.021937	0.018808	0.015381	0.018391	0.027354
HiTec	0.013768	0.044862	0.029548	0.024083	0.057499	0.031667	0.025756	0.017235	0.008714	0.029358
Telcm	0.014496	0.034589	0.022723	0.021937	0.031667	0.033226	0.020159	0.014624	0.009883	0.024191
Shops	0.013540	0.033981	0.021782	0.018808	0.025756	0.020159	0.024673	0.012810	0.008339	0.023347
Hlth	0.011173	0.020272	0.015728	0.015381	0.017235	0.014624	0.012810	0.019989	0.008842	0.016772
Utils	0.010333	0.014249	0.012443	0.018391	0.008714	0.009883	0.008339	0.008842	0.020717	0.012212
Other	0.016647	0.043447	0.028728	0.027354	0.029358	0.024191	0.023347	0.016772	0.012212	0.034396

데이터 확인 및 분석

- Asset_num = 자산의 개수(산업군의 개수) (10)

Annualized
- mu = 평균 수익률 * 12
- sigma = 수익률의 분산 * 12

Part 2 Portfolio Optimization

Import data: “10_Industry_Portfolios”

Portfolio Mean and Variance

```
[114]  1 # Equally-wegihted portfolio (1/N portfolio)
       2 eqw = np.ones(asset_num) / asset_num
       3 eqw
```

```
array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1])
```

```
[140]  1 # portfolio mean, variance, and standard deviation
       2 pfo_mu = np.dot(mu, eqw)
       3 pfo_var = np.dot(np.dot(eqw.T, sigma), eqw)
       4 pfo_var2 = np.dot(eqw.T, np.dot(sigma, eqw))
       5 pfo_std = np.sqrt(pfo_var)
       6 print("%0.5f, %0.5f, %0.5f, %0.5f" % (pfo_mu, pfo_var, pfo_var2, pfo_std))
```

```
0.09263, 0.02291, 0.02291, 0.15135
```

```
[164]  1 # Monthly mean and volatility
       2 a = pfo_mu/12
       3 b = pfo_std/np.sqrt(12)
       4 print(f"return: {a:.5f}")
       5 print(f"risk: {b:.5f}")
```

```
return: 0.00772
risk: 0.04369
```

- 동일 가중 포트폴리오
- 포트폴리오 수익률 = 평균 수익률 mu x weights
- 포트폴리오 분산 = weights x sigma x weights

$$\sigma_p^2 = [w_1, w_2, \cdots, w_n] \times \begin{bmatrix} \sigma_1^2 & \sigma_{1,2} & \cdots & \sigma_{1,n} \\ \sigma_{2,1} & \sigma_2^2 & \cdots & \sigma_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n,1} & \sigma_{n,2} & \cdots & \sigma_n^2 \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

- 10_Industry_Portfolios의 동일 가중 포트폴리오의 월간 수익률은 0.008, 위험은 0.044

Part 2 Portfolio Optimization

Import data: “10_Industry_Portfolios”

Cvxpy

```
[150]  1 import cvxpy as cp
```

```
[159]  1 gmv_ns2 = cp.Variable(asset_num)
       2 objective = cp.Minimize(cp.quad_form(gmv_ns2, sigma))
       3 constraints = [cp.sum(gmv_ns2) == 1, gmv_ns2 >= 0]
       4 problem = cp.Problem(objective, constraints)
       5 problem.solve(solver=cp.ECOS)
       6 gmv_ns2 = gmv_ns2.value
       7 gmv_ns2
```

```
array([3.75345370e-01, 4.62992349e-10, 1.30281547e-09, 8.13791088e-10,
       2.51570674e-09, 4.03544095e-09, 8.68404929e-02, 2.46846831e-01,
       2.90967296e-01, 1.09860996e-09])
```

```
[163]  1 # GMV return and risk (no short)
       2 a = np.dot(mu.T, gmv_ns2)
       3 b = np.sqrt(np.dot(gmv_ns2.T, np.dot(sigma, gmv_ns2)))
       4 print("return: %0.5f" % a)
       5 print("risk: %0.5f" % b)
```

```
return: 0.10092
risk: 0.11320
```

- Cvxpy로 최적화 진행
- GVM을 구하기 위해 목적함수는 분산의 최소화를, 제약조건은 가중치의 합은 1, no short을 위해 0이상으로 설정한다.
- ECOS solver로 최적화 문제 해결

- (no short) GVM포트폴리오의 수익률: 0.101, 위험: 0.11

Part 2 Portfolio Optimization

Import data: “10_Industry_Portfolios”

Cvxpy

```
[150]  1 import cvxpy as cp
```

```
[159]  1 gmv_ns2 = cp.Variable(asset_num)
       2 objective = cp.Minimize(cp.quad_form(gmv_ns2, sigma))
       3 constraints = [cp.sum(gmv_ns2) == 1, gmv_ns2 >= 0]
       4 problem = cp.Problem(objective, constraints)
       5 problem.solve(solver=cp.ECOS)
       6 gmv_ns2 = gmv_ns2.value
       7 gmv_ns2
```

NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other	
3.75E-01	4.63E-10	1.30E-09	8.14E-10	2.52E-09	4.04E-09	8.68E-02	2.47E-01	2.91E-01	1.10E-09	
38%	0%	0%	0%	0%	0%	9%	25%	29%	0%	100%

```
[163]  1 # GMV return and risk (no short)
       2 a = np.dot(mu.T, gmv_ns2)
       3 b = np.sqrt(np.dot(gmv_ns2.T, np.dot(sigma, gmv_ns2)))
       4 print("return: %0.5f" % a)
       5 print("risk: %0.5f" % b)
```

```
return: 0.10092
risk: 0.11320
```

- 최적화 결과

NoDur 38%, Shops 9%, Hlth 25%, Utils 29%

가장 작은 분산을 가지는 포트폴리오

- 모든 산업군에 투자하는 Scenario를 살펴보기 위해 동일 가중 포트폴리오로 가정하고 진행

Part 3 Simulation

Part 3 Simulation (Import data: "10_Industry_Portfolios")

Monte carlo Simulation

Initial Amount $ 1000000 .00

```
[ ]  1 while 1:
     2   try:
     3     Initial_Amount = int(input("Initial_Amount: "))
     4     break
     5   except:
     6     print('\nPlease enter the number')
```

```
Initial_Amount: 1000
```

- 수익률로 인한 자산의 움직임을 보기 위해 초기 자산을 입력 받는다.

Simulation Period in Years 30

5
10
15
20
25
30
35

```
1 print("\nPlease enter 5~10")
2 while 1:
3   try:
4     Simulation_Period_in_Years = int(input("Simulation Period
5   except:
6     print('\nPlease enter the number')
7     continue
8
9   if Simulation_Period_in_Years < 5 or Simulation_Period_in_Years > 30:
10      print('\nPlease enter a number between 5 and 10')
11  else:
12    break
```

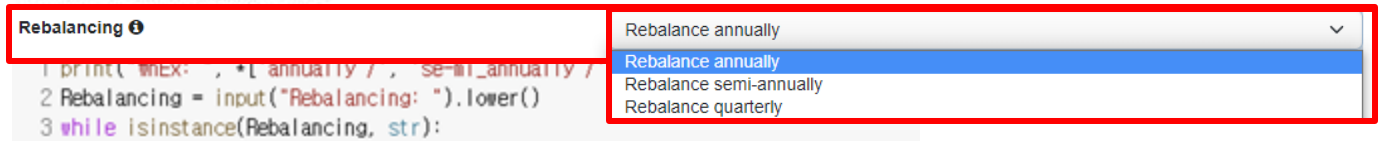
```
Please enter 5~10
Simulation Period in Years: 5
```

- 시뮬레이션을 통해 보고자 하는 기간을 입력한다.

Part 3 Simulation (Import data: "10_Industry_Portfolios")

```
print("\nPlease enter 100~1000")
while 1:
  try:
    number = int(input("Number of simulations: "))
  except:
    print('\nPlease enter the number')
    continue

  if number < 100 or number > 2000:
      print('\nPlease enter a number between 100 and 1000')
  else:
    break
```

```
Please enter 100~1000
Number of simulations: 1000
```




```
print('\nEx:  annually / se-mi_annually / quaterly / monthly')
Rebalancing = input("Rebalancing: ").lower()
while isinstance(Rebalancing, str):
  if Rebalancing in ['monthly', 'm', "1"]:
    period = 1
    break
  if Rebalancing in ['quaterly', 'q', "3"]:
    period = 3
    break
  elif Rebalancing in ['se-mi_annually', 'se-mi', "semi", 's', '6']:
    period = 6
    break
  elif Rebalancing in ['annually', 'a', "ann", '12']:
    period = 12
    break
  else:
    print("\nEnter as in the above")
    Rebalancing = input("Rebalancing: ").lower()
```

```
Ex:  annually / se-mi_annually / quaterly / monthly
Rebalancing: 3
```

- 시뮬레이션 횟수 설정
- Portfolio visualizer에는 X
- 100 ~ 2000사이 입력
- Print >> ~ 1000, 실행 시간 길어짐

- Rebalancing 주기 설정
- Annually, se-mi annually, quarterly + monthly

Part 3

Simulation (Import data: "10_Industry_Portfolios")

```
# Monte Carlo Method

mc_sims = number    # number of simulation
T = Simulation_Period_in_Years*12   # timeframe in days
#np.random.seed(0)    # for debugging

portfolio_sims_0 = np.full(shape=(T, mc_sims), fill_value=0.0) # not rebalancing
portfolio_sims_1 = np.full(shape=(T, mc_sims), fill_value=0.0) # 1 month rebalancing
portfolio_sims_r = np.full(shape=(T, mc_sims), fill_value=0.0) # r month rebalancing

initialPortfolio = int(Initial_Amount)

for m in range(0, mc_sims):
  # MC loops
  sample = np.random.multivariate_normal(meanReturns, covMatrix, size = T)
  x = sample + 1
  x[0] = x[0] * weights *initialPortfolio
  y = np.cumprod(x, axis=0)

  portfolio_sims_0[:, m ] = y.sum(axis=1) # not rebalancing
  portfolio_sims_1[:, m ] = np.cumprod(np.inner(weights, sample)+1)*initialPortfolio  # 1 month rebalancing
  portfolio_sims_r[:, m ] = portfolio_r(period) # r month rebalancing
```

- 시뮬레이션 코드
- Mc_sims에 시뮬레이션 횟수를, T에 시뮬레이션 기간을 initialPortfolio에는 초기 자산을 저장한다.
- 기간은 연(年) 단위로, 수익률은 월(月) 단위로 받아왔기 때문에 12를 곱해 T에 저장한다.

- [7~9] not rebalanced, 1 month, r month

결과를 저장할 array를 미리 선언한 뒤 for loop을 실행

- For loop은 입력 받은 횟수(mc_sims)만큼 진행한다.

Part 3

Simulation (Import data: "10_Industry_Portfolios")

Import data: "10_Industry_Portfolios"

```
for m in range(0, mc_sims):
    # MC loops
    sample = np.random.multivariate_normal(meanReturns, covMatrix, size = T)
    x = sample + 1
    x[0] = x[0] * weights *initialPortfolio
    y = np.cumprod(x, axis=0)

    portfolio_sims_0[:, m ] = y.sum(axis=1) # not rebalancing
```

```
np.random.seed(0)
print(f"mc_sims: {mc_sims}, T: {T}, initialPortfolio: {initialPortfolio}, seed: 0₩n");
sample = np.random.multivariate_normal(meanReturns, covMatrix, size = T)
df(sample, columns = asset_list)
```

mc_sims: 1000, T: 120, initialPortfolio: 1000, seed: 0

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
0	-0.055102	-0.091083	-0.091902	-0.046641	-0.081563	-0.104178	-0.077924	-0.051065	0.033033	-0.132696
1	-0.006712	-0.010999	-0.021426	0.089966	-0.017850	0.009134	-0.032573	0.005056	0.016754	-0.014333
2	0.019384	0.204383	0.128726	0.204789	0.172030	0.080306	0.114645	0.074266	0.073672	0.104248
3	0.009762	-0.043036	-0.009113	0.015992	0.018158	0.042845	0.015309	0.037225	-0.045658	-0.002487
4	0.101870	0.091156	0.072321	-0.044975	0.075989	0.045962	0.086586	0.050416	0.080299	0.057520
...	...	...	...	...	...	...	...	...	...	...
115	-0.006805	0.072834	-0.022790	-0.060955	0.011450	0.022501	0.023092	-0.016760	0.011429	0.025211
116	0.007574	0.045570	0.011407	0.122801	0.006600	0.032721	-0.002021	0.017638	0.045032	-0.006448
117	-0.024140	0.055280	-0.006340	-0.035319	0.035444	-0.018220	0.005087	-0.016530	-0.011711	-0.044012
118	-0.037450	-0.147590	-0.086707	-0.096922	-0.122475	-0.063691	-0.040347	-0.022660	-0.058102	-0.061188
119	0.028236	-0.004274	0.036563	0.077369	0.051548	0.043210	-0.017155	-0.007192	0.015141	-0.007454

120 rows × 10 columns

- Sample은 수익률을 random sampling한 matrix
- Np.random.multivariate_normal을 사용해 다변량 정규분포에서 값을 추출
- 함수에 사용할 parameter로 불러온 meanReturns(데이터의 평균수익률)과 covMatrix(공분산 행렬)을 넣고, size를 T로 설정
- 그 결과 아래와 같이 기간 120(month) x 10(산업군) 행렬이 생성됨
- (설명을 위해 random seed는 0으로 설정하고 진행)

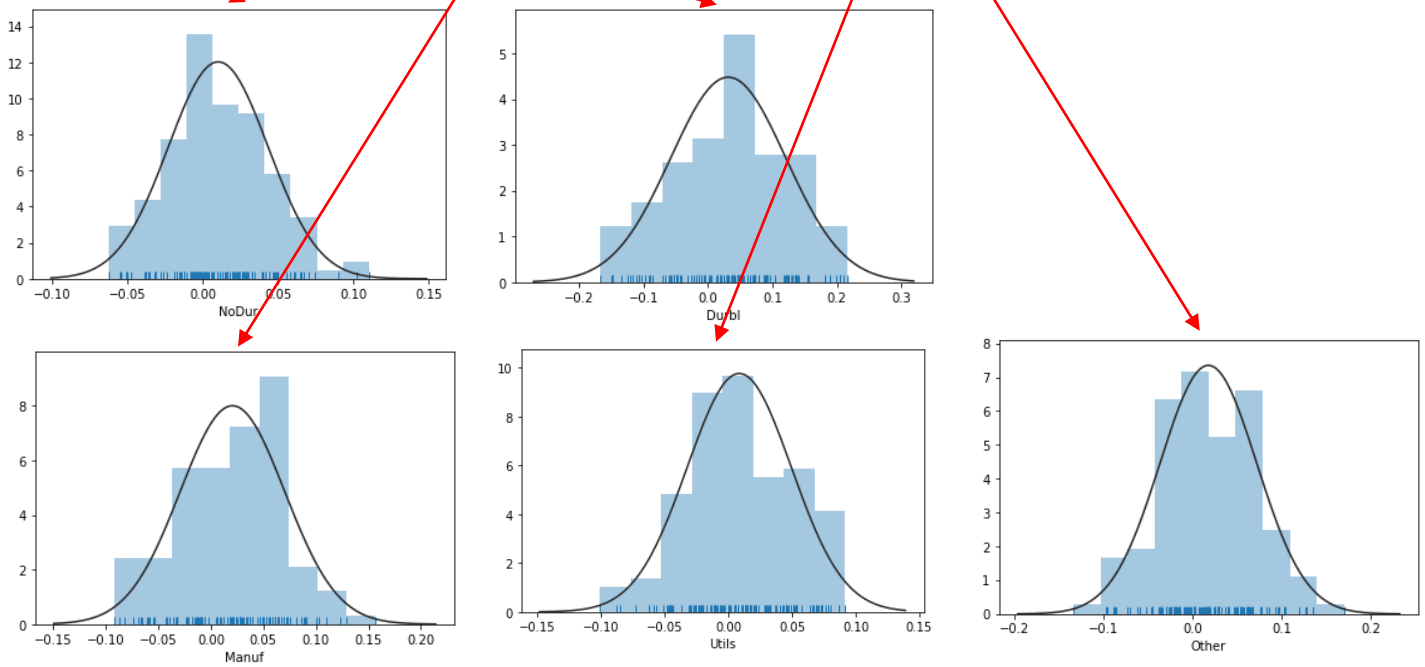
Part 3

Simulation (Import data: "10_Industry_Portfolios")

mc_sims: 1000, T: 120, initialPortfolio: 1000, seed: 0

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
0	-0.055102	-0.091083	-0.091902	0.046641	-0.081563	-0.104178	-0.077924	-0.051065	0.033033	-0.132696
1	-0.006712	-0.010999	-0.021426	0.089966	-0.017850	0.009134	-0.032573	0.005056	0.016754	-0.014333
2	0.019384	0.204383	0.128726	0.204789	0.172030	0.080306	0.114645	0.074266	0.073672	0.104248
3	0.009762	-0.043036	-0.009113	0.015992	0.018158	0.042845	0.015309	0.037225	-0.045658	-0.002487
4	0.101870	0.091156	0.072321	0.044975	0.075989	0.045962	0.086586	0.050416	0.080299	0.057520
...	...	...	...	...	...	...	...	...	...	...
115	-0.006805	0.072834	-0.022790	0.060955	0.011450	0.022501	0.023092	-0.016760	0.011429	0.025211
116	0.007574	0.045570	0.011407	0.122801	0.006600	0.032721	-0.002021	0.017638	0.045032	-0.006448
117	-0.024140	0.055280	-0.006340	-0.035319	0.035444	-0.018220	0.005087	-0.016530	-0.011711	-0.044012
118	-0.037450	-0.147590	-0.086707	0.096922	-0.122475	-0.063691	-0.040347	-0.022660	-0.058102	-0.061188
119	0.028236	-0.004274	0.036563	0.077369	0.051548	0.043210	-0.017155	-0.007192	0.015141	-0.007454

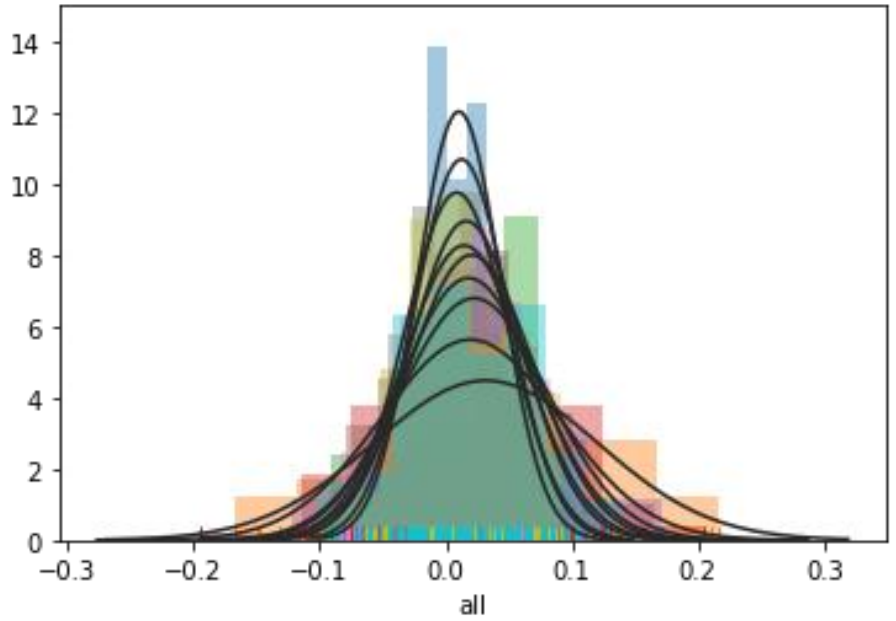
120 rows × 10 columns



- 아래 그래프는 sample matrix의 산업별 수익률에 대해 그린 histogram 그래프

sns.distplot(rug = True, kde = False, fit = sp.stats.norm)

- 모든 산업군의 수익률을 한번에 표현



Part 3 Simulation (Import data: "10_Industry_Portfolios")

mc_sims: 1000, T: 120, initialPortfolio: 1000, seed: 0

```
for m in range(0, mc_sims):
  # MC loops
  sample = np.random.multivariate_normal(meanReturns, covMatrix, size = T)
  x = sample + 1
  x[0] = x[0] * weights *initialPortfolio
  y = np.cumprod(x, axis=0)
```

- Sample은 수익률 데이터이기 때문에 자산해 곱해주기 위해 +1을 한 matrix 'x'를 만들어 준다.

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
0	-0.055102	-0.091083	-0.091902	-0.046641	-0.081563	-0.104178	-0.077924	-0.051065	0.033033	-0.132696
1	-0.006712	-0.010999	-0.021426	0.089966	-0.017850	0.009134	-0.032573	0.005056	0.016754	-0.014333
2	0.019384	0.204383	0.128726	0.204789	0.172030	0.080306	0.114645	0.074266	0.073672	0.104248
3	0.009762	-0.043036	-0.009113	0.015992	0.018158	0.042845	0.015309	0.037225	-0.045658	-0.002487
4	0.101870	0.091156	0.072321	-0.044975	0.075989	0.045962	0.086586	0.050416	0.080299	0.057520
...	...	...	...	...	...	...	...	...	...	...
115	-0.006805	0.072834	-0.022790	-0.060955	0.011450	0.022501	0.023092	-0.016760	0.011429	0.025211
116	0.007574	0.045570	0.011407	0.122801	0.006600	0.032721	-0.002021	0.017638	0.045032	-0.006448
117	-0.024140	0.055280	-0.006340	-0.035319	0.035444	-0.018220	0.005087	-0.016530	-0.011711	-0.044012
118	-0.037450	-0.147590	-0.086707	-0.096922	-0.122475	-0.063691	-0.040347	-0.022660	-0.058102	-0.061188
119	0.028236	-0.004274	0.036563	0.077369	0.051548	0.043210	-0.017155	-0.007192	0.015141	-0.007454

120 rows × 10 columns

+1

dataframe: x

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
0	0.961415	0.980011	0.938853	0.921902	0.980206	0.934530	1.029624	0.954702	0.963581	0.944837
1	0.947072	0.880164	0.951701	0.960957	0.950062	0.987746	0.928985	0.998876	1.025339	0.951104
2	0.982479	0.976297	0.942299	0.915916	0.957048	0.935177	0.966997	0.953984	0.958724	0.940247
3	0.951907	0.907839	1.009420	1.041212	0.986196	1.004350	0.974353	0.974947	1.009556	0.986117
4	0.965876	0.991463	0.957810	0.926737	0.978723	0.930188	1.004433	0.974880	0.960765	0.975526
...	...	...	...	...	...	...	...	...	...	...
115	1.087898	1.092358	1.139702	1.249382	1.114056	1.135255	1.061762	1.102405	1.019608	1.117501
116	1.006531	1.066895	0.997946	1.053878	1.051583	0.987445	1.011050	1.022327	1.007332	0.987640
117	1.016018	0.967731	1.017395	0.924470	0.972828	0.963181	0.994367	1.037251	0.986501	0.984511
118	1.033480	0.994555	1.005333	1.014498	1.011677	0.998366	1.029764	1.049428	1.051628	1.026948
119	0.954336	0.969267	0.996974	0.954132	0.975396	0.956652	0.951646	0.989616	1.011151	0.970320

120 rows × 10 columns

owerPoint

Part 3 Simulation (Import data: "10_Industry_Portfolios")

mc_sims: 1000, T: 120, initialPortfolio: 1000, seed: 0

```
12 for m in range(0, mc_sims):
13   # MC loops
14   sample = np.random.multivariate_normal(meanReturns, covMatrix, size = T)
15   x = sample + 1
16   x[0] = x[0] * weights *initialPortfolio
17   y = np.cumprod(x, axis=0)
```

- 첫 달의 수익률인 첫 행과 initialPortfolio인 1000과 가중치인 '1/10'을 곱해 수정한다.
- 동일 가중 포트폴리오의 첫 달의 자산 현황(수익률)을 만들기 위함

dataframe: x

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
0	0.961415	0.980011	0.938853	0.921902	0.980206	0.934530	1.029624	0.954702	0.963581	0.944837
1	0.947072	0.880164	0.951701	0.960957	0.950062	0.987746	0.928985	0.998876	1.025339	0.951104
2	0.982479	0.976297	0.942299	0.915916	0.957048	0.935177	0.966997	0.953984	0.958724	0.940247
3	0.951907	0.907839	1.009420	1.041212	0.986196	1.004350	0.974353	0.974947	1.009556	0.986117
4	0.965876	0.991463	0.957810	0.926737	0.978723	0.930188	1.004433	0.974880	0.960765	0.975526
...	...	...	...	...	...	...	...	...	...	...
115	1.087898	1.092358	1.139702	1.249382	1.114056	1.135255	1.061762	1.102405	1.019608	1.117501
116	1.006531	1.066895	0.997946	1.053878	1.051583	0.987445	1.011050	1.022327	1.007332	0.987640
117	1.016018	0.967731	1.017395	0.924470	0.972828	0.963181	0.994367	1.037251	0.986501	0.984511
118	1.033480	0.994555	1.005333	1.014498	1.011677	0.998366	1.029764	1.049428	1.051628	1.026948
119	0.954336	0.969267	0.996974	0.954132	0.975396	0.956652	0.951646	0.989616	1.011151	0.970320

120 rows × 10 columns

[첫 행] x [0.1, 0.1, … , 0.1] x 1000

initialPortfolio: 1000, weights: 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
'row 0' x weights x initialPortfolio

	0	1	2	3	4	5	6	7	8	9
0	96.141493	98.001062	93.885333	92.190190	98.020604	93.453049	102.962361	95.470152	96.358066	94.483660
1	0.947072	0.880164	0.951701	0.960957	0.950062	0.987746	0.928985	0.998876	1.025339	0.951104
2	0.982479	0.976297	0.942299	0.915916	0.957048	0.935177	0.966997	0.953984	0.958724	0.940247
3	0.951907	0.907839	1.009420	1.041212	0.986196	1.004350	0.974353	0.974947	1.009556	0.986117
4	0.965876	0.991463	0.957810	0.926737	0.978723	0.930188	1.004433	0.974880	0.960765	0.975526
...	...	...	...	...	...	...	...	...	...	...
115	1.087898	1.092358	1.139702	1.249382	1.114056	1.135255	1.061762	1.102405	1.019608	1.117501
116	1.006531	1.066895	0.997946	1.053878	1.051583	0.987445	1.011050	1.022327	1.007332	0.987640
117	1.016018	0.967731	1.017395	0.924470	0.972828	0.963181	0.994367	1.037251	0.986501	0.984511
118	1.033480	0.994555	1.005333	1.014498	1.011677	0.998366	1.029764	1.049428	1.051628	1.026948
119	0.954336	0.969267	0.996974	0.954132	0.975396	0.956652	0.951646	0.989616	1.011151	0.970320

120 rows × 10 columns

Part 3 Simulation (Import data: "10_Industry_Portfolios")

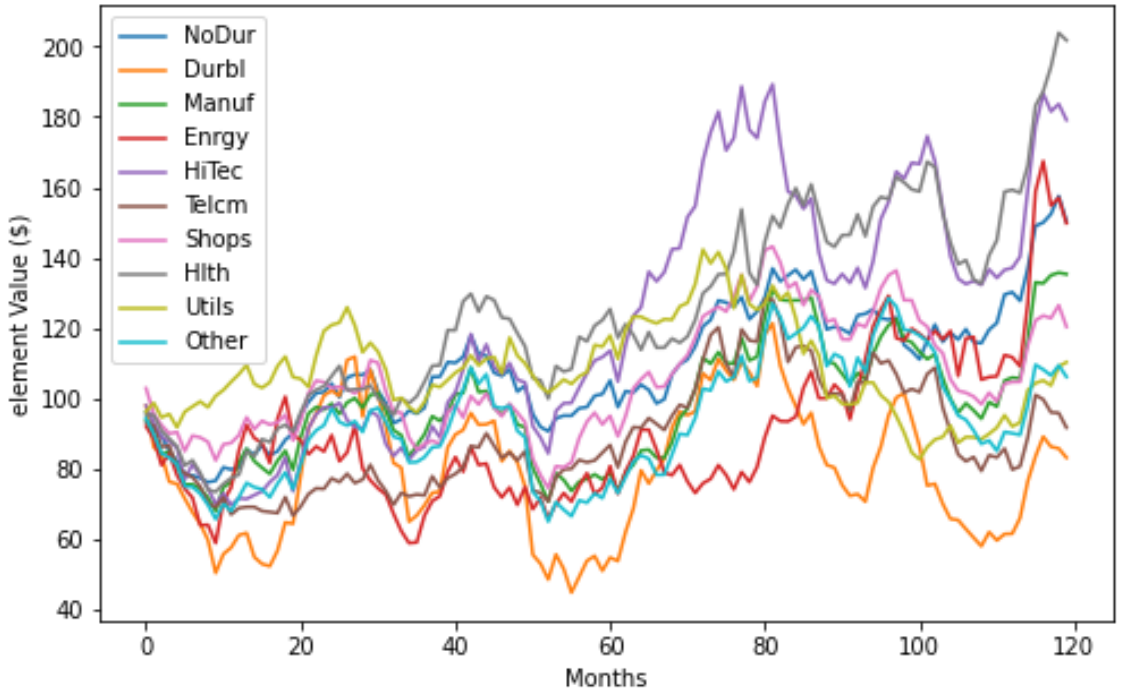
Rebalancing: not rebalanced / 1month / r month

```
for m in range(0, mc_sims):
  # MC loops
  sample = np.random.multivariate_normal(meanReturns, covMatrix, size = T)
  x = sample + 1
  x[0] = x[0] * weights *initialPortfolio

  y = np.cumprod(x, axis=0) # not rebalancing
  portfolio_sims_0[:, m ] = y.sum(axis=1)
```

- **not rebalanced** / 1month / r month
- Np.cumprod(axis=0)을 통해 새로 방향으로 누적 곱
- 그 결과 왼쪽과 같은 행렬의 얻게 되고
- 포트폴리오 구성(산업군)에 대한 가치 추이에 대해 그래프를 그리면 오른쪽과 같이 나오게 된다.

	0	1	2	3	4	5	6	7	8	9
0	96.141493	98.001062	93.885333	92.190190	98.020604	93.453049	102.962361	95.470152	96.358066	94.483660
1	91.052956	86.257052	89.350798	88.590791	93.125661	92.307886	95.650494	95.362822	98.799642	89.863747
2	89.457608	84.212535	84.195182	81.141708	89.125773	86.324235	92.493715	90.974586	94.721566	84.494154
3	85.155361	76.451385	84.988299	84.485724	87.895505	86.699751	90.121520	88.695440	95.626759	83.321143
4	82.249533	75.798685	81.402630	78.296065	86.025324	80.647033	90.520984	86.467391	91.874887	81.281908
...	...	...	...	...	...	...	...	...	...	...
115	149.061014	83.658864	133.029509	158.896566	177.456881	101.067938	122.230958	183.207392	104.430718	109.639288
116	150.034588	89.255233	132.756264	167.457568	186.610608	99.799002	123.581646	187.297793	105.196420	108.284122
117	152.437854	86.375050	135.065576	154.809523	181.540114	96.124488	122.885561	194.274888	103.776332	106.606892
118	157.541405	85.904753	135.785930	157.054022	183.660034	95.967455	126.543164	203.877556	109.134137	109.479786
119	150.347451	83.264607	135.375101	149.850321	179.141281	91.807416	120.424311	201.760422	110.351121	106.230406

120 rows × 10 columns



Part 3 Simulation (Import data: "10_Industry_Portfolios")

Rebalancing: not rebalanced / 1month / r month

```
for m in range(0, mc_sims):
  # MC loops
  sample = np.random.multivariate_normal(meanReturns, covMatrix, size = T)
  x = sample + 1
  x[0] = x[0] * weights *initialPortfolio

  y = np.cumprod(x, axis=0) # not rebalancing
  portfolio_sims_0[:, m ] = y.sum(axis=1)
```

```
y = np.cumprod(x, axis=0)
```

	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
0	96.141493	98.001062	93.885333	92.190190	98.020604	93.453049	102.962361	95.470152	96.358066	94.483660
1	91.052956	86.257052	89.350798	88.590791	93.125661	92.307886	95.650494	95.362822	98.799642	89.863747
2	89.457608	84.212535	84.195182	81.141708	89.125773	86.324235	92.493715	90.974586	94.721566	84.494154
3	85.155361	76.451385	84.988299	84.485724	87.895505	86.699751	90.121520	88.695440	95.626759	83.321143
4	82.249533	75.798685	81.402630	78.296065	86.025324	80.647033	90.520984	86.467391	91.874887	81.281908
...	...	...	...	...	...	...	...	...	...	...
115	149.061014	83.658864	133.029509	158.896566	177.456881	101.067938	122.230958	183.207392	104.430718	109.639288
116	150.034588	89.255233	132.756264	167.457568	186.610608	99.799002	123.581646	187.297793	105.196420	108.284122
117	152.437854	86.375050	135.065576	154.809523	181.540114	96.124488	122.885561	194.274888	103.776332	106.606892
118	157.541405	85.904753	135.785930	157.054022	183.660034	95.967455	126.543164	203.877556	109.134137	109.479786
119	150.347451	83.264607	135.375101	149.850321	179.141281	91.807416	120.424311	201.760422	110.351121	106.230406

120 rows × 10 columns

- 포트폴리오의 가치

Sum(axis = 1) 가로 방향 더하기 진행

- 하나의 시나리오에 대한 결과 생성 및 저장

```
df(portfolio_sims_0[:,0])
```

	0
0	930.087715
1	932.227816
2	1041.876037
3	1045.295080
4	1107.065890
...	...
115	8765.781863
116	9032.839666
117	9144.934390
118	8220.545420
119	8368.407285

120 rows × 1 columns

Part 3 Simulation (Import data: "10_Industry_Portfolios")

Rebalancing: not rebalanced / 1month / r month

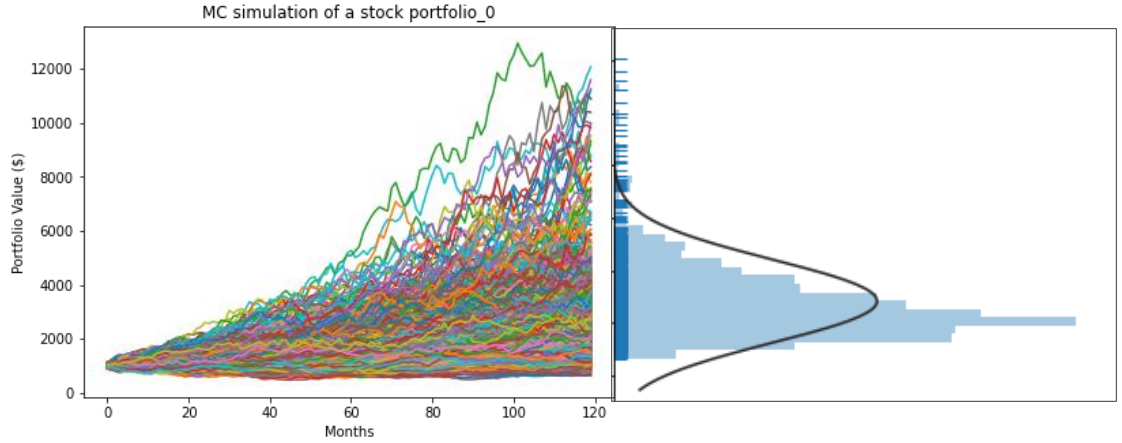
```
16  x[0] = x[0] * weights *initialPortfolio
17
18  y = np.cumprod(x, axis=0) # not rebalancing
19  portfolio_sims_0[:, m ] = y.sum(axis=1)
```

```
df(portfolio_sims_0)
```

	0	1	2	3	...	990	991	992	
0	930.087715	1039.713084	1001.695524	987.400363	...	1049.684227	989.332536	1047.219243	917
1	932.227816	1025.048808	1038.088174	994.604318	...	960.400877	1086.071963	963.476663	887
2	1041.876037	1014.505548	1026.338762	1063.368901	...	929.332126	1123.325356	1081.582333	895
3	1045.295080	983.346881	1035.353240	1152.171155	...	914.566443	1139.476271	1099.832607	822
4	1107.065890	889.087043	1018.036349	1161.396919	...	932.869036	1039.535285	1080.870737	837
...	...	...	...	...	...	...	...	...	
115	8765.781863	2208.590045	4359.908660	3880.449723	...	3450.789868	1514.829307	1940.914111	3932
116	9032.839666	2479.082886	4209.915520	4232.865603	...	3570.017761	1539.719681	1956.477279	4103
117	9144.934390	2362.974729	4349.914113	4448.959004	...	3429.769045	1574.334355	1851.676468	4143
118	8220.545420	2445.072819	4370.380490	4203.131532	...	3123.174994	1534.708835	1971.96	
119	8368.407285	2463.263636	4325.274716	4109.326632	...	3348.902485	1540.394550	2112.04	

120 rows × 1000 columns

- 전체 matrix
- 120개월, 1000개의 시나리오에 대한 그래프



	0	1	2	3	...	997	998	999
0	8368.0	2463.0	4325.0	4109.0	...	4399.0	2706.0	1329.0

1 rows × 1000 columns

	count	mean	std	min	25%	50%	75%	max
Not rebalanced	1000.0	2819.926	1580.538	624.21	1816.405	2420.523	3406.82	12071.008

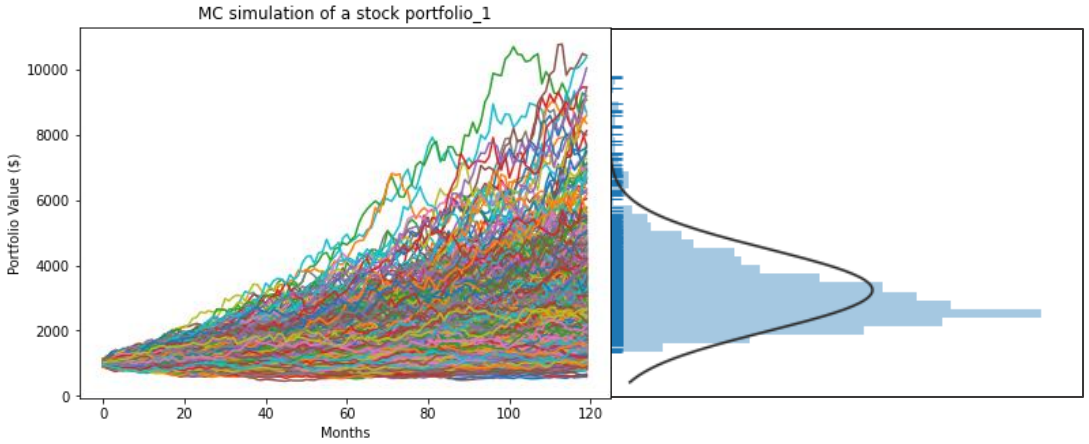
Part 3

Simulation (Import data: "10_Industry_Portfolios")

Rebalancing: not rebalanced / 1month / r month

```
for m in range(0, mc_sims):
  # MC loops
  sample = np.random.multivariate_normal(meanReturns, covMatrix, size = T)
  x = sample + 1
  x[0] = x[0] * weights *initialPortfolio
  y = np.cumprod(x, axis=0)

  portfolio_sims_0[:, m ] = y.sum(axis=1) # not rebalancing
  portfolio_sims_1[:, m ] = np.cumprod(np.inner(weights, sample)+1)*initialPortfolio # 1
  portfolio_sims_r[:, m ] = portfolio_r(period) # r month rebalancing
```

- not rebalanced / **1month** / r month
- Cumprod, inner 함수 이용




```
df(portfolio_sims_1)
```

	0	1	2	3	...	990	991
0	930.087715	1039.713084	1001.695524	987.400363	...	1049.684227	989.332536
1	931.670417	1025.515583	1038.139127	994.803589	...	960.195058	1085.937237
2	1041.276623	1014.029946	1026.788031	1063.183964	...	928.582168	1123.673676
3	1045.337237	984.007929	1035.857207	1152.464859	...	912.881200	1139.587912
4	1109.849550	889.681945	1018.329152	1162.113950	...	931.858174	1040.527928
...	...	...	...	...	...	...	...
115	7346.422238	2286.176353	4448.061337	3702.342289	...	3518.560974	1550.452503
116	7552.764901	2573.988523	4275.093518	3993.334370	...	3651.226505	1578.081766
117	7507.099711	2443.753523	4426.041992	4206.607937	...	3511.528548	1617.177984
118	6953.728611	2527.264382	4452.587428	3934.761417	...	3194.099811	1573.083121
119	7103.922853	2547.924394	4420.639077	3908.524985	...	3430.243386	1572.074239

120 rows × 1000 columns

	0	1	2	3	...	996	997	998	999
0	7104.0	2548.0	4421.0	3909.0	...	2169.0	3746.0	2649.0	1368.0

1 rows × 1000 columns

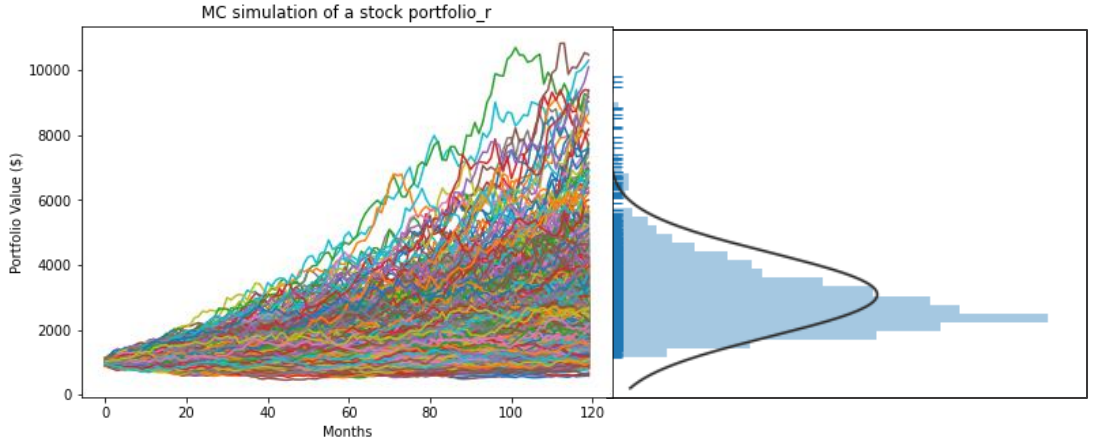
	count	mean	std	min	25%	50%	75%	max
1 month	1000.0	2776.75	1463.511	575.595	1811.945	2435.053	3356.887	10428.74

Part 3 Simulation (Import data: "10_Industry_Portfolios")

Rebalancing: not rebalanced / 1month / r month

```
for m in range(0, mc_sims):
  # MC loops
  sample = np.random.multivariate_normal(meanReturns, covMatrix, size = T)
  x = sample + 1
  x[0] = x[0] * weights *initialPortfolio
  y = np.cumprod(x, axis=0)

  portfolio_sims_0[:, m ] = y.sum(axis=1) # not rebalancing
  portfolio_sims_1[:, m ] = np.cumprod(np.inner(weights, sample)+1)*initialPortfolio  # 1
  portfolio_sims_r[:, m ] = portfolio_r(period) # r month rebalancing
```

- r = rebalancing
- (가정) input = 3
- 3달 주기 리밸런싱 결과
- Portfolio_r(period) 함수 정의




```
df(portfolio_sims_r)
```

	0	1	2	3	...	992	993	994
0	930.087715	1039.713084	1001.695524	987.400363	...	1047.219243	917.322139	1040.099114
1	932.227816	1025.048808	1038.088174	994.604318	...	963.476663	887.354534	934.064842
2	1041.876037	1014.505548	1026.338762	1063.368901	...	1081.582333	895.666712	944.712077
3	1045.938988	984.469450	1035.403970	1152.665326	...	1099.402734	820.060960	991.603678
4	1110.074646	891.259908	1018.044029	1162.297490	...	1078.017965	834.197057	956.341048
...	...	...	...	...	...	...	...	...
115	7386.194802	2288.130637	4419.893404	3729.737051	...	1945.116343	3816.858252	3586.250393
116	7590.380210	2575.770510	4248.800679	4028.839538	...	1969.870106	3949.211032	3843.655495
117	7544.487592	2445.445348	4398.820783	4244.009344	...	1889.791479	3951.600730	3878.911301
118	6982.761552	2530.345926	4425.218440	3968.332729	...	2006.195256	3954.476831	4045.699964
119	7129.835824	2548.015729	4394.973613	3940.351791	...	2147.412435	4153.185772	4141.770774

120 rows × 1000 columns

	0	1	2	3	...	996	997	998	999
0	7130.0	2548.0	4395.0	3940.0	...	2183.0	3743.0	2648.0	1380.0

1 rows × 1000 columns

	count	mean	std	min	25%	50%	75%	max
r(3) month	1000.0	2778.227	1465.752	577.022	1814.163	2440.876	3364.247	10473.693

Part 3 Simulation (Import data: "10_Industry_Portfolios")

Rebalancing: not rebalanced / 1month / r month

```
20  portfolio_sims_0[:, m ] = y.sum(axis=1) # not
21  portfolio_sims_1[:, m ] = np.cumprod(np.inner
22  portfolio_sims_r[:, m ] = portfolio_r(period)
```

```
def portfolio_r(period):
  portfolio_sims_r = np.full(shape=(T), fill_value=0.0) # r month rebalancing
  tmp = copy.deepcopy(x)
  for i in range(T):
    if T % period ==0:
      if i == T-1:
        tmp[i-period+1:] = np.cumprod(tmp[i-period+1:], axis=0)
        portfolio_sims_r[i-period+1:] = tmp[i-period+1:].sum(axis=1)
      elif i%period == period-1:
        tmp[i-period+1:i+1] = np.cumprod(tmp[i-period+1:i+1], axis=0)
        portfolio_sims_r[i-period+1:i+1] = tmp[i-period+1:i+1].sum(axis=1)
        tmp[i+1] = tmp[i+1] * weights * portfolio_sims_r[i]

    else:
      if T-i < period and i%period ==0:
        tmp[i:] = np.cumprod(tmp[i:], axis=0)
        portfolio_sims_r[i:] = tmp[i:].sum(axis=1)
        break
      elif i % period ==(period-1):
        tmp[i-period+1:i+1] = np.cumprod(tmp[i-period+1:i+1], axis=0)
        portfolio_sims_r[i-period+1:i+1] = tmp[i-period+1:i+1].sum(axis=1)
        tmp[i+1] = tmp[i+1] * weights * portfolio_sims_r[i]
  return portfolio_sims_r
```

- Argument: Period << (input)Rebalancing
- (가정)Period = 3

Part 3 Simulation (Import data: "10_Industry_Portfolios")

Rebalancing: not rebalanced / 1month / r month

```
20  portfolio_sims_0[:, m ] = y.sum(axis=1) # not
21  portfolio_sims_1[:, m ] = np.cumprod(np.inner
22  portfolio_sims_r[:, m ] = portfolio_r(period)
```

```
1  def portfolio_r (3):
2    portfolio_sims_r = np.full(shape=(T), fill_value=0.0)   # 3 month rebalancing
3    tmp = copy.deepcopy(x)
4    for i in range(T):
5      if T % 3 ==0:
6        if i == T-1:
7          tmp[i-2:] = np.cumprod(tmp[i-2:], axis=0)
8          portfolio_sims_r[i-2:,m] = tmp[i-2:].sum(axis=1)
9        elif i%3 ==2:
10         tmp[i-2:i+1] = np.cumprod(tmp[i-2:i+1], axis=0)
11         portfolio_sims_r[i-2:i+1,m] = tmp[i-2:i+1].sum(axis=1)
12         tmp[i+1] = tmp[i+1] * weights * portfolio_sims_r[i,m]
13
14     else:
15       if T-i < 3 and i%2 ==0:
16         tmp[i:] = np.cumprod(tmp[i:], axis=0)
17         portfolio_sims_r[i:,m] = tmp[i:].sum(axis=1)
18         break
19       elif i % 3 ==2:
20         tmp[i-2:i+1] = np.cumprod(tmp[i-2:i+1], axis=0)
21         portfolio_sims_r[i-2:i+1,m] = tmp[i-2:i+1].sum(axis=1)
22         tmp[i+1] = tmp[i+1] * weights * portfolio_sims_r[i,m]
23   return portfolio_sims_r[:,m]
```

- (가정)Period = 3
- [2-3] portfolio의 각 기간별로 자산 가치를 넣을 T(12개월)길이의 array 생성
- x를 copy한 'tmp 'array를 생성

(x를 직접 사용할 경우 sampling 된 수익률 matrix를 직접 수정하기 때문에 다른 작업에 영향

- T(120)이 period(3)의 배수일 경우와 아닐 경우로 구분

Part 3 Simulation (Import data: "10_Industry_Portfolios")

Rebalancing: not rebalanced / 1month / r month

```
20  portfolio_sims_0[:, m ] = y.sum(axis=1) # not
21  portfolio_sims_1[:, m ] = np.cumprod(np.inner
22  portfolio_sims_r[:, m ] = portfolio_r(period)
```

```
1  def portfolio_r (3):
2    portfolio_sims_r = np.full(shape=(T), fill_value=0.0)   # 3 month rebalancing
3    tmp = copy.deepcopy(x)
4    for i in range(T):
5      if T % 3 ==0:
6        if i == T-1:
7          tmp[i-2:] = np.cumprod(tmp[i-2:], axis=0)
8          portfolio_sims_r[i-2:,m] = tmp[i-2:].sum(axis=1)
9        elif i%3 ==2:
10         tmp[i-2:i+1] = np.cumprod(tmp[i-2:i+1], axis=0)
11         portfolio_sims_r[i-2:i+1,m] = tmp[i-2:i+1].sum(axis=1)
12         tmp[i+1] = tmp[i+1] * weights * portfolio_sims_r[i,m]
13
14     else:
15       if T-i < 3 and i%2 ==0:
16         tmp[i:] = np.cumprod(tmp[i:], axis=0)
17         portfolio_sims_r[i:,m] = tmp[i:].sum(axis=1)
18         break
19       elif i % 3 ==2:
20         tmp[i-2:i+1] = np.cumprod(tmp[i-2:i+1], axis=0)
21         portfolio_sims_r[i-2:i+1,m] = tmp[i-2:i+1].sum(axis=1)
22         tmp[i+1] = tmp[i+1] * weights * portfolio_sims_r[i,m]
23   return portfolio_sims_r[:,m]
```

- (가정)Period = 3
- T(120)이 period(3)의 배수일 경우와 아닐 경우로 구분
- 3의 배수 일 때 동작

- [5-12] (배수일 경우) T가 3이면 (0부터 시작해서, 3번째 행이면) 1행부터 3행까지 column 별로 곱하고[10], 가로 방향으로 더한 뒤 만들어 놓은 portfolio_sims_r에 저장 (0~3행이 모두 같이 더해져 저장)
- 다음 행(4)에는, 직전에 더해진 마지막 portfolio의 값에 weights를 곱한 뒤, 4번째 행의 수익률과 곱해줍니다.
- 이렇게 반복하다 마지막 T가 119번 째가 되면, 마지막 반복을 실행합니다.

- [14-22] (배수가 아닐 경우)
- 이때도 기본 포맷은 동일한데, 마지막 행의 개수가 3개 이하 (rebalancing 주기 이하)이기 때문에, 이를 위해 구분

Part 3

Simulation (Import data: "10_Industry_Portfolios")

Rebalancing: not rebalanced / 1month / r month

Month	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other	sum
1	94.84485	90.09483	90.13872	95.38846	94.79546	92.93145	92.45276	91.535	102.2291	86.78489	931.1955
2	0.985253	0.981545	0.998854	1.091466	0.992092	1.012032	0.980798	1.007225	1.010805	0.967892	
3	1.039962	1.19836	1.10904	1.202161	1.179305	1.090585	1.111043	1.041219	1.059692	1.132379	
4	1.015752	0.952746	1.005486	1.013496	1.008768	1.036699	1.03017	1.033101	0.959469	1.004104	
5	1.096808	1.105708	1.075022	0.957121	1.061359	1.052996	1.076706	1.038058	1.086402	1.060368	
6	1.045304	1.048591	1.063497	1.072568	1.056914	1.046545	1.053517	1.055955	1.005318	1.056492	
7	1.046822	1.033266	1.063824	1.017385	1.064793	1.019053	1.065196	1.029766	0.988579	1.071859	
8	0.968959	0.965444	0.98325	1.000524	0.988281	0.949693	0.97208	0.957844	0.950083	0.982064	
9	1.040372	1.08068	1.044602	1.140204	1.090758	1.028211	1.046038	1.086624	0.99547	1.043185	
10	1.032219	1.020966	1.009242	1.099941	1.002433	1.016089	1.038216	1.021216	1.052742	1.007461	

Month	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other	sum
2	93.44617	88.43213	90.03542	104.1133	94.04582	94.0496	90.67749	92.19634	103.3337	83.9984	934.3283
3	97.18047	105.9735	99.85288	125.1609	110.9087	102.5691	100.7466	95.99658	109.5019	95.11803	**1043.01**
rebalanced	104.3009	104.3009	104.3009	104.3009	104.3009	104.3009	104.3009	104.3009	104.3009	104.3009	**1043.01**
4	105.9438	99.37223	104.8731	105.7085	105.2154	108.1286	107.4476	107.7533	100.0734	104.7289	1049.245
5	116.2	109.8767	112.7408	101.1758	111.6713	113.859	115.6895	111.8542	108.72	111.0512	1112.839
6	121.4643	115.2157	119.8996	108.518	118.0269	119.1586	121.8809	118.113	109.2982	117.3247	**1168.9**
rebalanced	116.89	116.89	116.89	116.89	116.89	116.89	116.89	116.89	116.89	116.89	**1168.9**
7	122.363	120.7784	124.3504	118.9221	124.4636	119.1171	124.5107	120.3693	115.555	125.2896	1215.719
8	118.5647	116.6048	122.2675	118.9844	123.005	113.1247	121.0344	115.295	109.7868	123.0424	1181.71

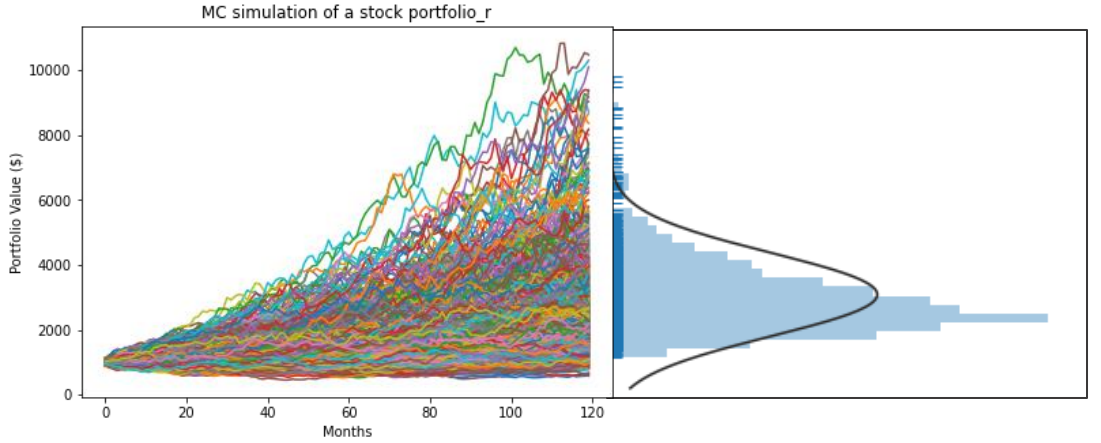
- 앞의 과정을 Excel로 표현
  - 위는 수익률(sample(or x)) 데이터,
  - 아래는 rebalancing 과정

Part 3 Simulation (Import data: "10_Industry_Portfolios")

Rebalancing: not rebalanced / 1month / r month

```
for m in range(0, mc_sims):
  # MC loops
  sample = np.random.multivariate_normal(meanReturns, covMatrix, size = T)
  x = sample + 1
  x[0] = x[0] * weights *initialPortfolio
  y = np.cumprod(x, axis=0)

  portfolio_sims_0[:, m ] = y.sum(axis=1) # not rebalancing
  portfolio_sims_1[:, m ] = np.cumprod(np.inner(weights, sample)+1)*initialPortfolio  # 1
  portfolio_sims_r[:, m ] = portfolio_r(period) # r month rebalancing
```

- r = rebalancing
- (가정) input = 3
- 3달 주기 리밸런싱 결과
- Portfolio_r(period) 함수 정의




```
df(portfolio_sims_r)
```

	0	1	2	3	...	992	993	994
0	930.087715	1039.713084	1001.695524	987.400363	...	1047.219243	917.322139	1040.099114
1	932.227816	1025.048808	1038.088174	994.604318	...	963.476663	887.354534	934.064842
2	1041.876037	1014.505548	1026.338762	1063.368901	...	1081.582333	895.666712	944.712077
3	1045.938988	984.469450	1035.403970	1152.665326	...	1099.402734	820.060960	991.603678
4	1110.074646	891.259908	1018.044029	1162.297490	...	1078.017965	834.197057	956.341048
...	...	...	...	...	...	...	...	...
115	7386.194802	2288.130637	4419.893404	3729.737051	...	1945.116343	3816.858252	3586.250393
116	7590.380210	2575.770510	4248.800679	4028.839538	...	1969.870106	3949.211032	3843.655495
117	7544.487592	2445.445348	4398.820783	4244.009344	...	1889.791479	3951.600730	3878.911301
118	6982.761552	2530.345926	4425.218440	3968.332729	...	2006.195256	3954.476831	4045.699964
119	7129.835824	2548.015729	4394.973613	3940.351791	...	2147.412435	4153.185772	4141.770774

120 rows × 1000 columns

	0	1	2	3	...	996	997	998	999
0	7130.0	2548.0	4395.0	3940.0	...	2183.0	3743.0	2648.0	1380.0

1 rows × 1000 columns

	count	mean	std	min	25%	50%	75%	max
r(3) month	1000.0	2778.227	1465.752	577.022	1814.163	2440.876	3364.247	10473.693

Part 4 Performance Evaluation

Part 4

Performance Evaluation

(EQW) Portfolio - Return, Risk, Sharpe Ratio

Return

```
[260] 1 #mean(arithmetic)
      2 meanReturns_w = np.dot(meanReturns, weights)
      3 print(f"mean: {round(meanReturns_w*100, 1)}%")
      4
      5 #annualized(arithmetic)
      6 meanReturns_ann = meanReturns_w*12
      7 print(f"annualized: {round(meanReturns_ann, 2)*100}%")
      8
      9 # of months
     10 print("number of months: ", len(returns))
     11
     12 # of years
     13 print("number of years: ", round(len(returns)/12, 2))
```

```
mean: 0.8%
annualized: 10.0%
number of months:  250
number of years:  20.83
```

Risk

```
[253] 1 vol = np.sqrt(weights@covMatrix@weights.T)
      2 vol_a = vol * (12**(1/2))
      3 print(('volatility: %.5f' % vol))
      4 print("annualized vol: %.4f" % vol_a)
```

```
volatility: 0.04434
annualized vol: 0.1536
```

Sharpe Ratio

```
[49]  1 RF_rate = 0.001 # per month
      2 SR_m = (meanReturns_w - RF_rate)/vol
      3 print("SR (monthly): %.5f" % SR_m)
      4
      5 SR_a = (meanReturns_ann - RF_rate*12) / vol_a
      6 print("SR (annualy)): %.5f" % SR_a)
```

```
SR (monthly): 0.16741
SR (annualy)): 0.57992
```

- (동일 가중) 포트폴리오에 대한 수익률, 위험, 샤프지수
- 수익률, 위험은 앞과 동일
- Mean: 0.08% (month) / 10.0% (year)
- risk: 0.044 (month) / 0.15 (year)

Sharpe ratio computes the expected excess return of a portfolio for every unit of volatility it takes

$$\text{Sharpe ratio} = \frac{E(r) - r_f}{\sigma(r)}$$

- 무 위험 수익률: 0.001 (per month)
- Sharpe ratio: 0.17 (month) / 0.58 (year)

Part 4

Performance Evaluation

Simulation data– Return (balance)

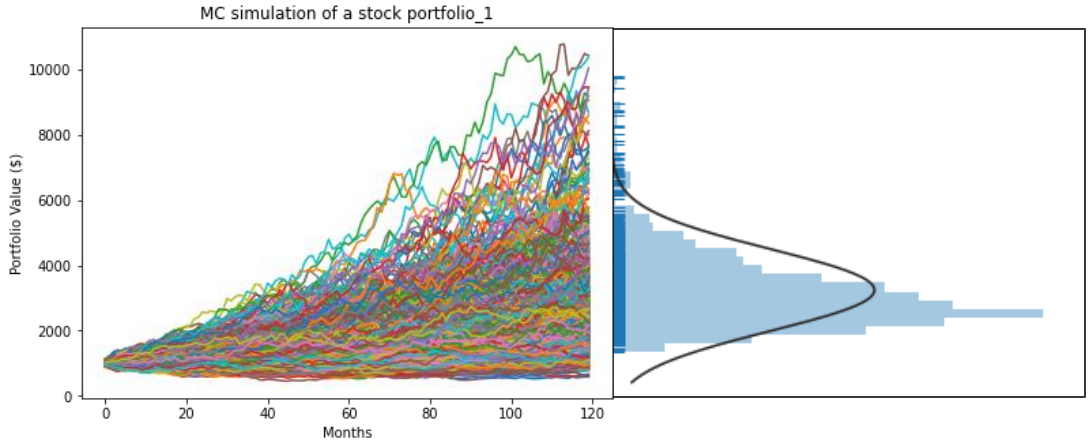
```
for m in range(0, mc_sims):
  # MC loops
  sample = np.random.multivariate_normal(meanReturns, covMatrix, size = T)
  x = sample + 1
  x[0] = x[0] * weights *initialPortfolio
  y = np.cumprod(x, axis=0)

  portfolio_sims_0[:, m ] = y.sum(axis=1) # not rebalancing
  portfolio_sims_1[:, m ] = np.cumprod(np.inner(weights, sample)+1)*initialPortfolio # 1
  portfolio_sims_r[:, m ] = portfolio_r(period) # r month rebalancing
```

```
df(portfolio_sims_1)
```

	0	1	2	3	...	990	991
0	930.087715	1039.713084	1001.695524	987.400363	...	1049.684227	989.332536
1	931.670417	1025.515583	1038.139127	994.803589	...	960.195058	1085.937237
2	1041.276623	1014.029946	1026.788031	1063.183964	...	928.582168	1123.673676
3	1045.337237	984.007929	1035.857207	1152.464859	...	912.881200	1139.587912
4	1109.849550	889.681945	1018.329152	1162.113950	...	931.858174	1040.527928
...	...	...	...	...	...	...	...
115	7346.422238	2286.176353	4448.061337	3702.342289	...	3518.560974	1550.452503
116	7552.764901	2573.988523	4275.093518	3993.334370	...	3651.226505	1578.081766
117	7507.099711	2443.753523	4426.041992	4206.607937	...	3511.528548	1617.177984
118	6953.728611	2527.264382	4452.587428	3934.761417	...	3194.099811	1573.083121
119	7103.922853	2547.924394	4420.639077	3908.524985	...	3430.243386	1572.074239

120 rows × 1000 columns

- 1달 주기 리밸런싱 결과



	0	1	2	3	...	996	997	998	999
0	7104.0	2548.0	4421.0	3909.0	...	2169.0	3746.0	2649.0	1368.0

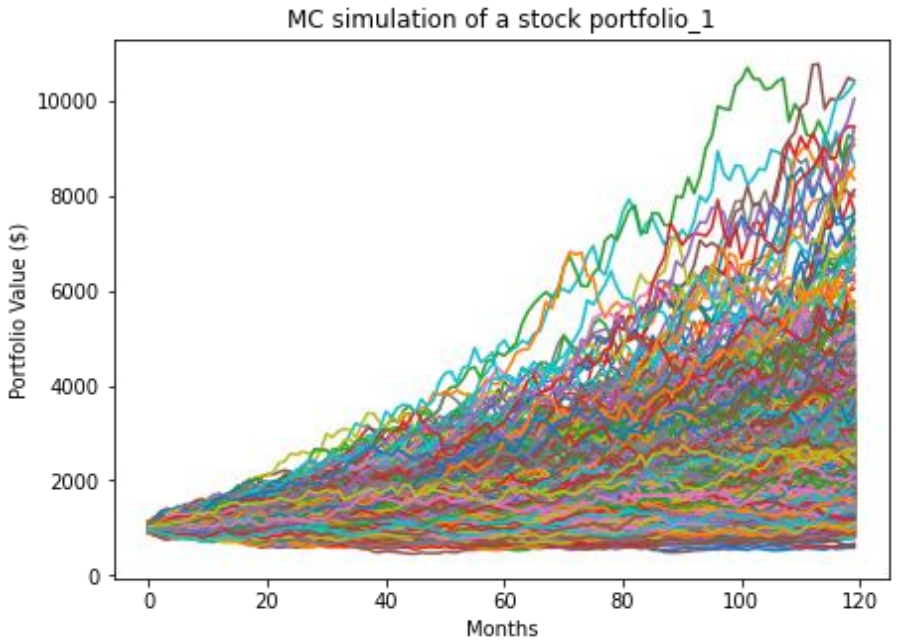
1 rows × 1000 columns

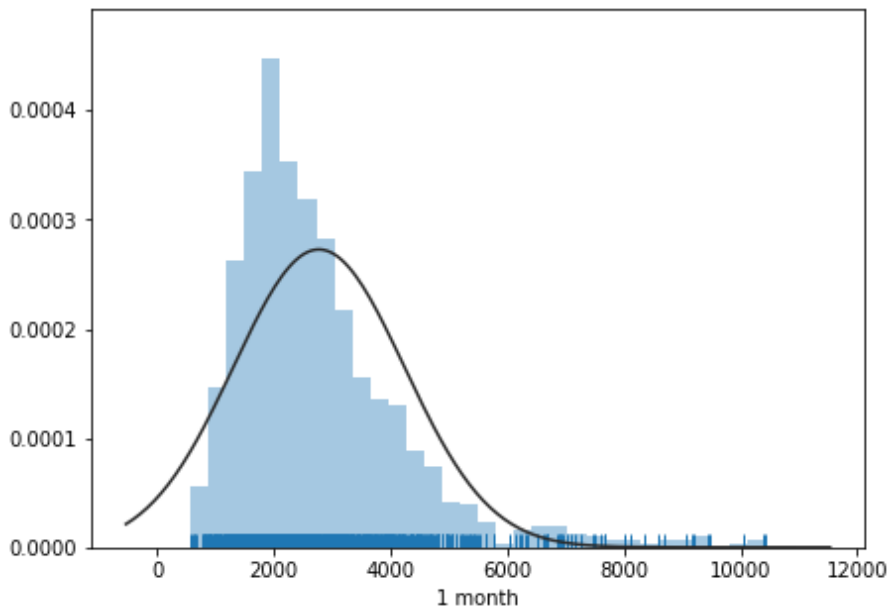
	count	mean	std	min	25%	50%	75%	max
1 month	1000.0	2776.75	1463.511	575.595	1811.945	2435.053	3356.887	10428.74

Part 4

Performance Evaluation

Simulation data (1 month rebalanced) - Return(balance)






```
1 df_balance.describe()
```

	balance
count	1000.000000
mean	2776.750218
std	1463.511295
min	575.595106
25%	1811.945217
50%	2435.052716
75%	3356.886657
max	10428.739822

- 1달 주기 리밸런싱 결과
- 개수: 1000 (시뮬레이션 횟수)
- 평균: 2,777 $
- 표준편차: 1,463 $
- 최대값: 10,429 $
- 최소값: 576 $

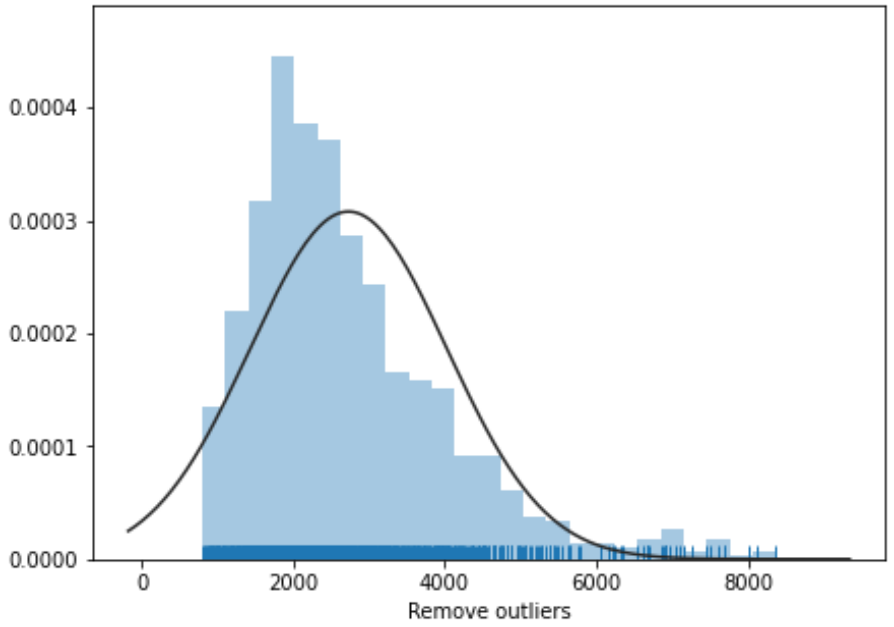
Part 4

Performance Evaluation

Simulation data (1 month rebalanced) - Return(balance)

```
# Remove outliers
alpha = 0.01
low = df_balance.balance.quantile(alpha)
print("low:", low)
high = df_balance.balance.quantile(1 - alpha)
print("high:", high, "₩n")
ret_trim = df()
ret_trim = df_balance[(df_balance.balance > low) & (df_balance.balance < high)]
print(df_balance.min(), df_balance.max(), ret_trim.min(), ret_trim.max(), sep = '₩n')
```

```
low: 805.9830171402947
high: 8358.846966508389

balance    575.595106
dtype: float64
balance    10428.739822
dtype: float64
balance    806.004105
dtype: float64
balance    8356.404204
dtype: float64
```




```
df(ret_trim).describe().T
```

	count	mean	std	min	25%	50%	75%	max
balance	980.0	2729.725232	1296.517567	806.004105	1825.05443	2435.052716	3332.974667	8356.404204

- Outliers제거
- 상, 하위 1% (1000 * 1% = 10)
- 데이터 개수: 980 (1000 – 10*2)
- 평균: 2730
- 표준편차: 1297
- 최대값: 8356
- 최소값: 806

Part 4

Performance Evaluation

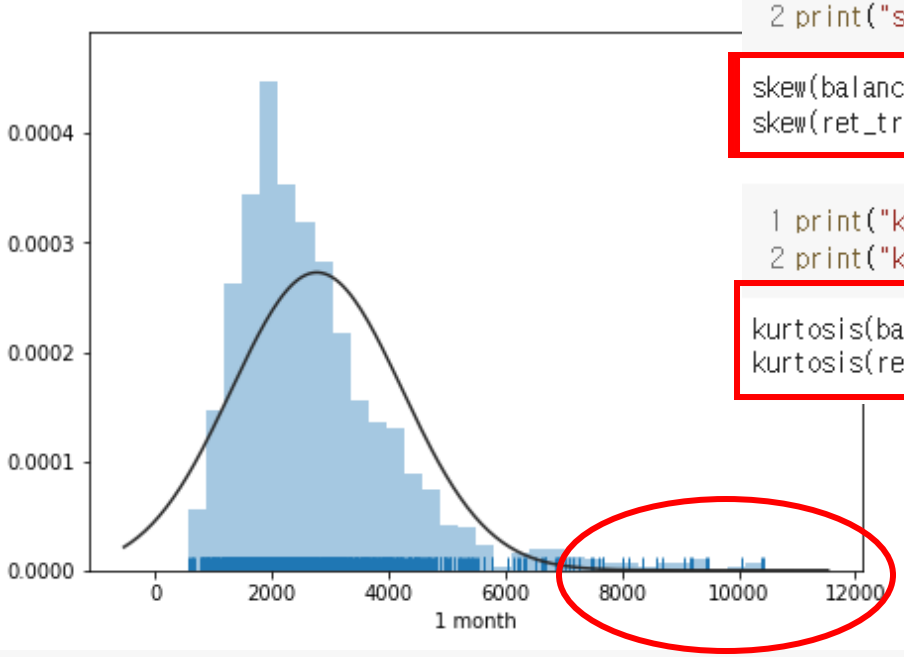
Simulation data (1 month rebalanced) - Return(balance)

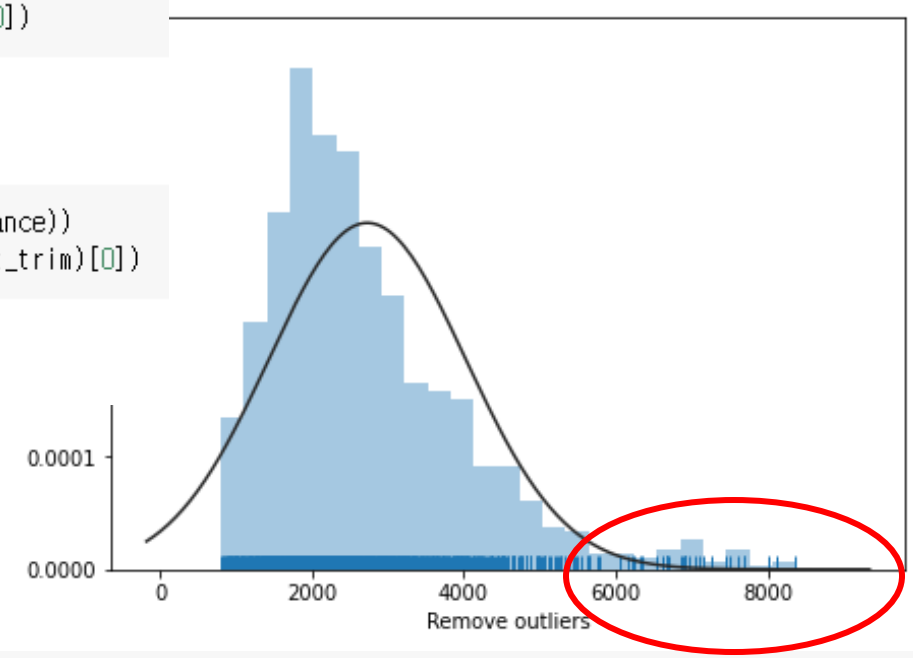
```
1 print("skew(balance):", sp.stats.skew(balance))
2 print("skew(ret_trim):", sp.stats.skew(ret_trim)[0])
```

```
skew(balance): 1.7754195762039102
skew(ret_trim): 1.326588246147105
```

```
1 print("kurtosis(balance):", sp.stats.kurtosis(balance))
2 print("kurtosis(ret_trim):", sp.stats.kurtosis(ret_trim)[0])
```

```
kurtosis(balance): 4.60971516812428
kurtosis(ret_trim): 2.1736693663107 26
```






```
1 df_balance.describe().T
```

	count	mean	std	min	25%	50%	75%	max
balance	1000.0	2776.750218	1463.511295	575.595106	1811.945217	2435.052716	3356.886657	10428.739822

```
1 df(ret_trim).describe().T
```

	count	mean	std	min	25%	50%	75%	max
balance	980.0	2729.725232	1296.517567	806.004105	1825.05443	2435.052716	3332.974667	8356.404204

- 두 데이터 비교
- Outliers를 제거한 오른쪽 그래프가 더 적은 극 값, 짧은 꼬리를 가짐

- 왼쪽 그래프(before) 의 skewness: 1.78
- 오른쪽 그래프(after) 의 skewness: 1.33
- 둘 다 positive skew이기 때문에 평균이 중앙값보다 큰, 오른쪽들의 값들이 더 많은 형태

- Kurtosis는 극 값에 대한 척도
- 왼쪽 그래프 >> 극 값 제거
- kurtosis: 왼쪽 4.61 > 오른쪽: 2.17 -->> 값이 낮아짐

Part 4

Performance Evaluation

Simulation data (1 month rebalanced) - MDD

```
mdd_all_1 = []
for s in range(portfolio_sims_1.shape[1]):
    mdd_value = mdd(portfolio_sims_1, s)[0]
    mdd_all_1.append(mdd_value)
```

```
def mdd (portfolio_sims, m):
    dd = df({'price': portfolio_sims[:, m]})
    dd = df(dd, columns = ['price', 'Highest past', 'Drawdown'])
    dd.iloc[0, 1] = dd.iloc[0, 0]

    for i in range(len(dd)):
        dd.iloc[i, 1] = dd['price'][:i+1].max()
        dd.iloc[i, 2] = dd['price'][i]/dd['Highest past'][i] - 1

    mdd = dd['Drawdown'].min()
    idx = dd['Drawdown'].idxmin()
    return [mdd, idx]
```

- 1달 주기 리밸런싱 결과
- Mdd 함수 정의
- portfolio_sims_1.shape[1]은 시나리오 개로 1000을 의미

```
portfolio_sims_0.shape[1]
```

```
1000
```

Part 4

Performance Evaluation

Simulation data (1 month rebalanced) - MDD

```python
def mdd (portfolio_sims, m):
    dd = df({'price': portfolio_sims[:, m]})
    dd = df(dd, columns = ['price', 'Highest past', 'Drawdown'])
    dd.iloc[0, 1] = dd.iloc[0, 0]

    for i in range(len(dd)):
        dd.iloc[i, 1] = dd['price'][:i+1].max()
        dd.iloc[i, 2] = dd['price'][i]/dd['Highest past'][i] - 1

    mdd = dd['Drawdown'].min()
    idx = dd['Drawdown'].idxmin()
    return [mdd, idx]
```

- Dd = drawdown
- Portfolio_sims_1과 m(시나리오 인덱스)을 인자로 받아
- Portfolio_sims에 저장된 각각의 시나리오에 대해 drawdown을 계산한다.

```python
dd = df({'price': portfolio_sims_1[:, 0]})
dd = df(dd, columns = ['price', 'Highest past', 'Drawdown'])
dd.iloc[0, 1] = dd.iloc[0, 0]
dd
```

	price	Highest past	Drawdown
0	930.087715	930.087715	NaN
1	931.670417	NaN	NaN
2	1041.276623	NaN	NaN
3	1045.337237	NaN	NaN
4	1109.849550	NaN	NaN
...	...	...	...
115	7346.422238	NaN	NaN
116	7552.764901	NaN	NaN
117	7507.099711	NaN	NaN
118	6953.728611	NaN	NaN
119	7103.922853	NaN	NaN

120 rows × 3 columns

```python
mdd (portfolio_sims_1, 0)[2]
```

	price	Highest past	Drawdown
0	930.087715	930.087715	0.000000
1	931.670417	931.670417	0.000000
2	1041.276623	1041.276623	0.000000
3	1045.337237	1045.337237	0.000000
4	1109.849550	1109.849550	0.000000
...	...	...	...
115	7346.422238	7346.422238	0.000000
116	7552.764901	7552.764901	0.000000
117	7507.099711	7552.764901	-0.006046
118	6953.728611	7552.764901	-0.079314
119	7103.922853	7552.764901	-0.059428

120 rows × 3 columns

Part 4

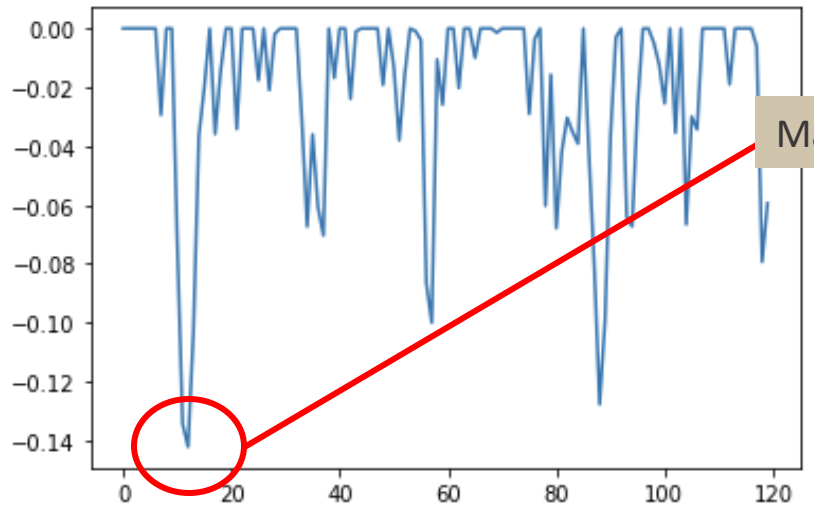
Performance Evaluation

Simulation data (1 month rebalanced) - M

```
def mdd (portfolio_sims, m):
  dd = df({'price': portfolio_sims[:, m]})
  dd = df(dd, columns = ['price', 'Highest past', 'Drawdown'])
  dd.iloc[0, 1] = dd.iloc[0, 0]

  for i in range(len(dd)):
    dd.iloc[i, 1] = dd['price'][:i+1].max()
    dd.iloc[i, 2] = dd['price'][i]/dd['Highest past'][i] - 1

  mdd = dd['Drawdown'].min()
  idx = dd['Drawdown'].idxmin()
  return [mdd, idx]
```

```
-0.142169005413198
```

```
[<matplotlib.lines.Line2D at 0x7fea9061f940>]
```



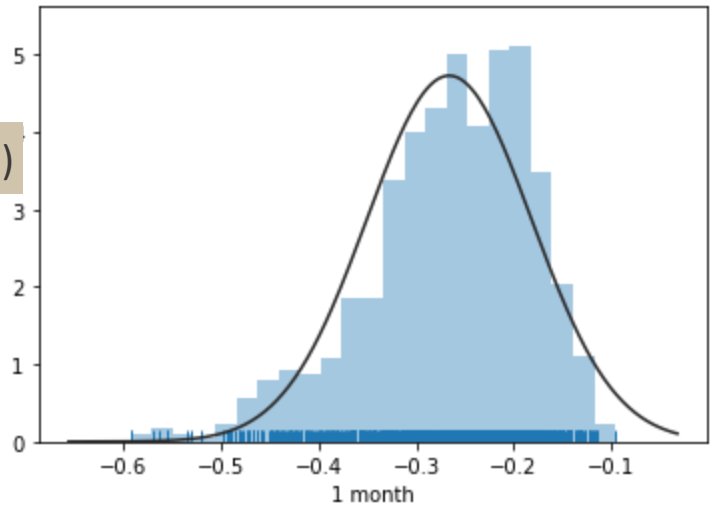

```
mdd (portfolio_sims_1, 0)[2].head(20)
```

	price	Highest past	Drawdown
0	930.087715	930.087715	0.000000
1	931.670417	931.670417	0.000000
2	1041.276623	1041.276623	0.000000
3	1045.337237	1045.337237	0.000000
4	1109.849550	1109.849550	0.000000
5	1164.378891	1164.378891	0.000000
6	1209.467323	1209.467323	0.000000
7	1173.936610	1209.467323	-0.029377
8	1239.904517	1239.904517	0.000000
9	1275.685010	1275.685010	0.000000
10	1183.543954	1275.685010	-0.072229
11	1104.240335	1275.685010	-0.134394
12	1094.322141	1275.685010	-0.142169
15	1250.018319	1275.685010	-0.020120
16	1287.349688	1287.349688	0.000000
17	1241.170462	1287.349688	-0.035872
18	1268.259217	1287.349688	-0.014829
19	1350.210204	1350.210204	0.000000

Maximal Drawdown in first scenario (index 0)

- 시나리오에 대해 drawdown을 계산하고 그 중 가장 작은(Maximal Drawdown)에 대해 그 값과 인덱스를 함수 mdd에 return한다.

- 이를 모든 시나리오에 대해 반복(1000) 함으로써 1 month rebalancing의 Maximal Drawdown의 분포를 구하게 된다.



Part 4

Performance Evaluation

Simulation data (1 month rebalanced) – Var, CVar

VAR

	0	1	2	3	...	996	997	998	999
0	7104.0	2548.0	4421.0	3909.0	...	2169.0	3746.0	2649.0	1368.0

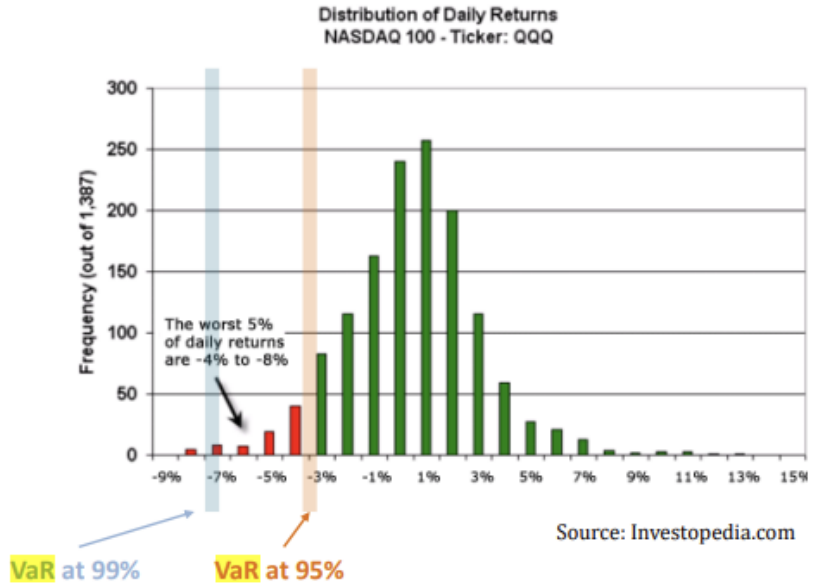
1 rows × 1000 columns

```
[65]  1 portResults = pd.Series(portfolio_sims_1[-1, :])
      2
      3 VaR = initialPortfolio - mcVaR(portResults, alpha = 5)
      4 CVaR = initialPortfolio - mcCVaR(portResults, alpha = 5)
      5
      6 print("VaR ${}".format(round(VaR, 2)))
      7 print("CVaR ${}".format(round(CVaR, 2)))
```

```
VaR $-101.43
CVaR $106.78
```

```
1 def mcVaR(returns, alpha = 5):
2   if isinstance(returns, pd.Series):
3     return np.percentile(returns, alpha)#, interpolation='nearest')
4   else:
5     raise TypeError("Expected a pandas data series")
6
7 def mcCVaR(returns, alpha = 5):
8   if isinstance(returns, pd.Series):
9     belowVaR = returns <= mcVaR(returns, alpha = alpha)
10    return returns[belowVaR].mean()
11  else:
12    raise TypeError("Expected a pandas data series")
```

Value-at-Risk



- VaR = value at risk
- find the worst-case loss of an investment (like MDD)
- Limitations:

VaR does not fully represent returns beyond the minimum loss

Part 4

Performance Evaluation

Simulation data (1 month rebalanced) – Var, CVar

VAR

	0	1	2	3	...	996	997	998	999
0	7104.0	2548.0	4421.0	3909.0	...	2169.0	3746.0	2649.0	1368.0

1 rows × 1000 columns

```
[65]  1 portResults = pd.Series(portfolio_sims_1[-1, :])
      2
      3 VaR = initialPortfolio - mcVaR(portResults, alpha = 5)
      4 CVaR = initialPortfolio - mcCVaR(portResults, alpha = 5)
      5
      6 print("VaR ${}".format(round(VaR, 2)))
      7 print("CVaR ${}".format(round(CVaR, 2)))
```

```
VaR $-101.43
CVaR $106.78
```

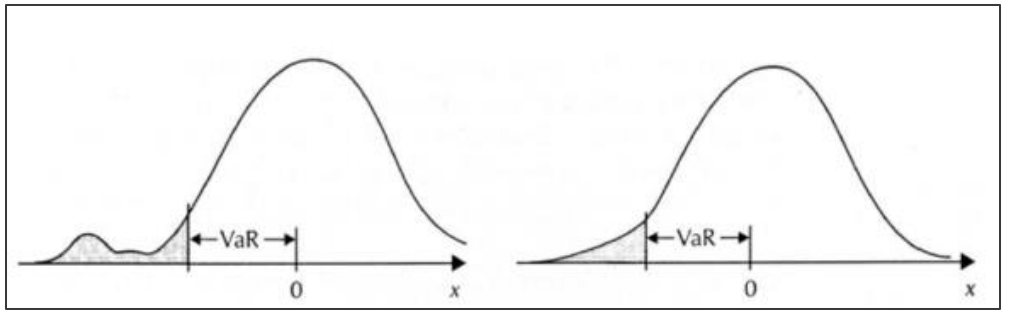
```
1 def mcVaR(returns, alpha = 5):
2   if isinstance(returns, pd.Series):
3     return np.percentile(returns, alpha)#, interpolation='nearest')
4   else:
5     raise TypeError("Expected a pandas data series")
6
7 def mcCVaR(returns, alpha = 5):
8   if isinstance(returns, pd.Series):
9     belowVaR = returns <= mcVaR(returns, alpha = alpha)
10    return returns[belowVaR].mean()
11  else:
12    raise TypeError("Expected a pandas data series")
```

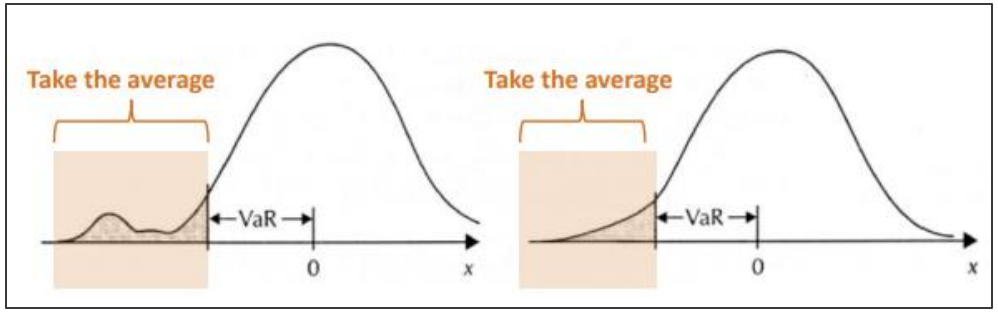
- Limitations:

VaR does not fully represent returns beyond the minimum loss

- Cvar = Conditional Value-at-Risk

computing the expected return of scenarios that are worse than the VaR level





Part 4

Performance Evaluation

Simulation data (1 month rebalanced) – Var, CVar

```python
def mcVaR(returns, alpha = 5):
  if isinstance(returns, pd.Series):
    return np.percentile(returns, alpha, interpolation='nearest')
  else:
    raise TypeError("Expected a pandas data series")

def mcCVaR(returns, alpha = 5):
  if isinstance(returns, pd.Series):
    belowVaR = returns <= mcVaR(returns, alpha = alpha)
    return returns[belowVaR].mean()
  else:
    raise TypeError("Expected a pandas data series")
```

- VaR = value at risk
- find the worst-case loss of an investment (like MDD)

1101.4276432898948

Idx	0
393	1076.316315
527	108(
278	108
197	1089.702313
24	1100.806496

Var = initialPortfolio - mcVaR(portResults, alpha = 5)

= 1000 – 1101.43 = -101.43

```python
VaR = initialPortfolio - mcVaR(portResults, alpha = 5)
CVaR = initialPortfolio - mcCVaR(portResults, alpha = 5)

print("VaR ${}".format(round(VaR, 2)))
print("CVaR ${}".format(round(CVaR, 2)))
```

```
VaR $-101.43
CVaR $106.78
```

Part 4

Performance Evaluation

Simulation data (1 month rebalanced) – Var, CVar

```
1 def mcVaR(returns, alpha = 5):
2   if isinstance(returns, pd.Series):
3     return np.percentile(returns, alpha)#, interpolation='nearest')
4   else:
5     raise TypeError("Expected a pandas data series")
6
7 def mcCVaR(returns, alpha = 5):
8   if isinstance(returns, pd.Series):
9     belowVaR = returns <= mcVaR(returns, alpha = alpha)
10    return returns[belowVaR].mean()
11  else:
12    raise TypeError("Expected a pandas data series")
```

- Cvar = Conditional Value-at-Risk
- computing the expected return of scenarios that are worse than the VaR level

```
1 mcCVaR(portResults, alpha = 5)
```

```
893.2231371726673
```

```
3 VaR = initialPortfolio - mcVaR(portResults, alpha = 5)
4 CVaR = initialPortfolio - mcCVaR(portResults, alpha = 5)
5
6 print("VaR ${}".format(round(VaR, 2)))
7 print("CVaR ${}".format(round(CVaR, 2)))
```

```
VaR $-101.43
CVaR $106.78
```

CVar = initialPortfolio - mcCVaR(portResults, alpha = 5)

= 1000 – 893.22 = 106.78

Part 5 Backtest

Part 5

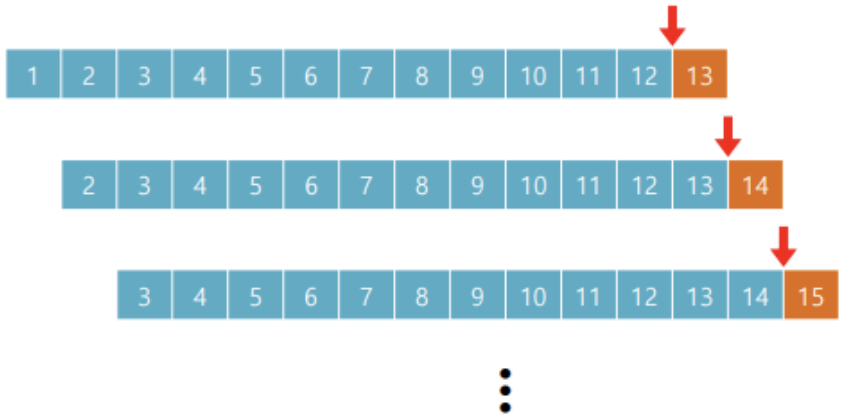
Backtest

“10_Industry_Portfolios” - Equally Weighted Portfolio

Backtest

- Rebalance period: Re-allocate (re-optimize) period
- Lookback period: Recent historical period used for estimating inputs of the model

- *Example*: 1-month rebalancing, 12-month lookback



1. Retrieve all necessary data

➢ 필요한 데이터는 포트폴리오를 구성했던 10_Industry_Portfolios 데이터로 진행합니다.

2. Run a for-loop to go through each rebalancing period

➢ Rebalance Period: 1-month

➢ Lookback Period: 12-month

➢ Total investment period: 2021.01. ~ 2021.12

3. In each for-loop iteration,

➢ Calculate inputs (mean, covariance, …) from lookback period

➢ Perform optimization

➢ Compute portfolio return during evaluation period

➢ Save any other values necessary (portfolio weight, …)

4. After the for-loop, run performance evaluation

Part 5

Backtest

“10_Industry_Portfolios” - Equally Weighted Portfolio

```
invest_period_s = "2021-01"
invest_period_e = "2021-12"
period_lb = 12

estimate = np.zeros((period_lb, 2))
backtest = np.zeros(period_lb)

for i in range(12):
  lookback = parse(invest_period_s) - relativedelta(months=period_lb+i)
  lookback_s = lookback.strftime("%Y-%m")
  lookback_e = parse(lookback_s) + relativedelta(months=period_lb-1)
  # space for optimization code used estimate
  train = returns.loc[lookback_s:lookback_e]
  estimate[i] = history(train, weights)

  backtest[i] = np.inner(returns.iloc[i], weights)
```

3. In each for-loop iteration,

- ➢ Calculate inputs (mean, covariance, …) from lookback period
- ➢ Perform optimization
- ➢ Compute portfolio return during evaluation period
- ➢ Save any other values necessary (portfolio weight, …)

- Rebalance Period: 1-month
- Lookback Period: 12-month
- Total investment period: 2021.01. ~ 2021.12

- Lookback period에 대한 estimate을 저장한 공간 ‘estimate’과 수익률에 대한 결과를 저장한 ‘backtest’를 생성

Part 5

Backtest

“10_Industry_Portfolios” - Equally Weighted Portfolio

```
invest_period_s = "2021-01"
invest_period_e = "2021-12"
period_lb = 12

estimate = np.zeros((period_lb, 2))
backtest = np.zeros(period_lb)

for i in range(12):
    lookback = parse(invest_period_s) - relativedelta(months=period_lb+i)
    lookback_s = lookback.strftime("%Y-%m")
    lookback_e = parse(lookback_s) + relativedelta(months=period_lb-1)
    # space for optimization code used estimate
    train = returns.loc[lookback_s:lookback_e]
    estimate[i] = history(train, weights)

    backtest[i] = np.inner(returns.iloc[i], weights)
```

- For loop은 Total investment period 만큼 반복 실행하고 rolling analysis 기법을 사용하기 때문에 그에 맞게 날짜를 수정해 나간다.
- 포트폴리오의 return matrix에서 lookback period만큼의 데이터 slicing 을 한 뒤 mean, cov 등을 추정하기 위해 history 함수에 대입한다.

```
lookback = parse(invest_period_s) - relativedelta(months=period_lb)
lookback_s = lookback.strftime("%Y-%m")
lookback_e = parse(lookback_s) + relativedelta(months=period_lb)
lookback_e = lookback_e.strftime("%Y-%m")
print(lookback_s, lookback_e)
```

```
2020-01 2021-01
```

Lookback Period

```
print(lookback_s, lookback_e)
print()
returns.loc[lookback_s:lookback_e]
```

```
2020-01 2021-01
```

Date	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
2020-01	-0.0038	0.0589	-0.0295	-0.1187	0.0331	-0.0199	0.0093	-0.0201	0.0483	-0.0137
2020-02	-0.0873	-0.0729	-0.0859	-0.1530	-0.0691	-0.0595	-0.0680	-0.0539	-0.0985	-0.0984
2020-03	-0.1149	-0.2276	-0.1679	-0.3449	-0.0973	-0.1337	-0.0760	-0.0500	-0.1301	-0.1887
2020-04	0.0801	0.2586	0.1094	0.3238	0.1525	0.0956	0.1804	0.1341	0.0507	0.1054
2020-05	0.0330	0.0722	0.0598	0.0052	0.0825	0.0478	0.0442	0.0405	0.0456	0.0388
2020-06	-0.0003	0.1431	0.0251	-0.0040	0.0602	-0.0252	0.0420	-0.0152	-0.0502	-0.0007
2020-07	0.0587	0.1843	0.0449	-0.0480	0.0694	0.0507	0.0951	0.0443	0.0637	0.0383
2020-08	0.0445	0.4019	0.0633	-0.0107	0.1050	0.0551	0.0816	0.0245	-0.0225	0.0721
2020-09	-0.0198	-0.0897	-0.0002	-0.1490	-0.0511	-0.0212	-0.0387	-0.0148	-0.0027	-0.0297
2020-10	-0.0256	-0.0329	-0.0076	-0.0453	-0.0182	-0.0385	-0.0257	-0.0442	0.0449	-0.0190
2020-11	0.1002	0.3385	0.1386	0.2846	0.1083	0.1443	0.0838	0.0952	0.0263	0.1576
2020-12	0.0500	0.1565	0.0260	0.0616	0.0496	0.0529	0.0149	0.0476	0.0063	0.0531
2021-01	-0.0411	0.1145	-0.0249	0.0464	0.0056	-0.0342	0.0000	0.0322	-0.0040	-0.0285

Part 5

Backtest

“10_Industry_Portfolios” - Equally Weighted Portfolio

```
1 def history (train):
2     mean = train.mean()
3     var = train.var()
4     std = np.sqrt(var)
5
6     return [mean, std]
```

```
1 print(lookback
2 print()
3 returns.loc[lo
```

2020-01 2021-01

Date	NoDur								Utils	Other
2020-01	-0.0038	0.0589	-0.0295	-0.1187	0.0331	-0.0199	0.0093	-0.0201	0.0483	-0.0137
2020-02	-0.0873	-0.0729	-0.0859	-0.1530	-0.0691	-0.0595	-0.0680	-0.0539	-0.0985	-0.0984
2020-03	-0.1149	-0.2276	-0.1679	-0.3449	-0.0973	-0.1337	-0.0760	-0.0500	-0.1301	-0.1887
2020-04	0.0801	0.2586	0.1094	0.3238	0.1525	0.0956	0.1804	0.1341	0.0507	0.1054
2020-05	0.0330	0.0722	0.0598	0.0052	0.0825	0.0478	0.0442	0.0405	0.0456	0.0388
2020-06	-0.0003	0.1431	0.0251	-0.0040	0.0602	-0.0252	0.0420	-0.0152	-0.0502	-0.0007
2020-07	0.0587	0.1843	0.0449	-0.0480	0.0694	0.0507	0.0951	0.0443	0.0637	0.0383
2020-08	0.0445	0.4019	0.0633	-0.0107	0.1050	0.0551	0.0816	0.0245	-0.0225	0.0721
2020-09	-0.0198	-0.0897	-0.0002	-0.1490	-0.0511	-0.0212	-0.0387	-0.0148	-0.0027	-0.0297
2020-10	-0.0256	-0.0329	-0.0076	-0.0453	-0.0182	-0.0385	-0.0257	-0.0442	0.0449	-0.0190
2020-11	0.1002	0.3385	0.1386	0.2846	0.1083	0.1443	0.0838	0.0952	0.0263	0.1576
2020-12	0.0500	0.1565	0.0260	0.0616	0.0496	0.0529	0.0149	0.0476	0.0063	0.0531
2021-01	-0.0411	0.1145	-0.0249	0.0464	0.0056	-0.0342	0.0000	0.0322	-0.0040	-0.0285

- Lookback period에 대해 mean, cov 등을 계산하기 위한 history 함수
- 전달 받은 기간 내의 수익률 데이터에 대해 평균과 표준편차를 반환한다.

Part 5

Backtest

“10_Industry_Portfolios” - Equally Weighted Portfolio

```
invest_period_s = "2021-01"
invest_period_e = "2021-12"
period_lb = 12

estimate = np.zeros((period_lb, 2))
backtest = np.zeros(period_lb)

for i in range(12):
  lookback = parse(invest_period_s) - relativedelta(months=period_lb+i)
  lookback_s = lookback.strftime("%Y-%m")
  lookback_e = parse(lookback_s) + relativedelta(months=period_lb-1)

  train = returns.loc[lookback_s:lookback_e]
  estimate[i] = history(train)
  # space for optimization code used estimate
  backtest[i] = np.inner(returns.iloc[i], weights)
```

3. In each for-loop iteration,

- Calculate inputs (mean, covariance, …) from lookback period
- Perform optimization
- Compute portfolio return during evaluation period
- Save any other values necessary (portfolio weight, …)

- 원래 포맷대로라면 학습 기간 내 얻은 추정치를 통해 최적화를 진행하고, weights를 구해 rebalancing에 반영하게 된다.
- 하지만 최초 모든 산업군에 투자하는 포트폴리오에 대해 시뮬레이션을 진행했기에,
- Bakctest에서도 동일한 동일 가중 포트폴리오로 test를 진행하기 위해 최적화 코드는 넣지 않았다.

Part 5

Backtest

“10_Industry_Portfolios” - Equally Weighted Portfolio

```
invest_period_s = "2021-01"
invest_period_e = "2021-12"
period_lb = 12

estimate = np.zeros((period_lb, 2))
backtest = np.zeros(period_lb)

for i in range(12):
  lookback = parse(invest_period_s) - relativedelta(months=period_lb+i)
  lookback_s = lookback.strftime("%Y-%m")
  lookback_e = parse(lookback_s) + relativedelta(months=period_lb-1)

  train = returns.loc[lookback_s:lookback_e]
  estimate[i] = history(train)
  # space for optimization code used estimate
  backtest[i] = np.inner(returns.iloc[i], weights)
```

3. In each for-loop iteration,

- Calculate inputs (mean, covariance, …) from lookback period
- Perform optimization
- Compute portfolio return during evaluation period
- Save any other values necessary (portfolio weight, …)

- 10개 산업군에 대한 동일 가중 포트폴리오이기 때문에 Weights는 [0.1] x 10의 array이다.
- 해당 weights를 통해 최초 불러왔던 ‘returns’ 테이블에서 포트폴리오의 수익률을 구한다.
- 구한 수익률은 ‘backtest’ array에 저장한다.

Part 5

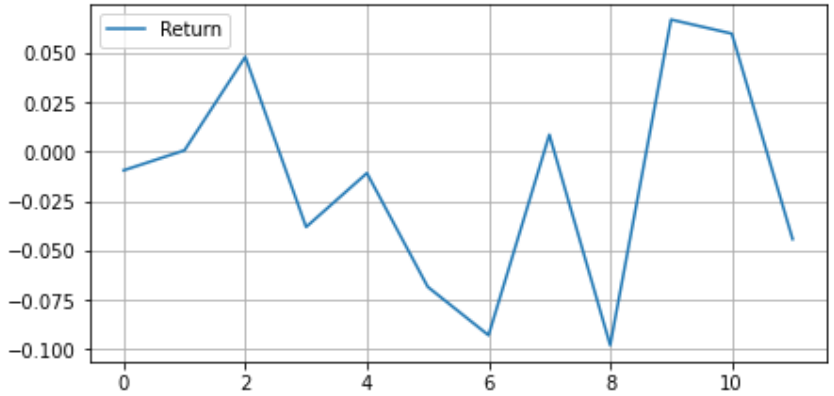
Backtest

“10_Industry_Portfolios” - Equally Weighted Portfolio

```
invest_period_s = "2021-01"
invest_period_e = "2021-12"
period_lb = 12

estimate = np.zeros((period_lb, 2))
backtest = np.zeros(period_lb)

for i in range(12):
  lookback = parse(invest_period_s) - relativedelta(months=period_lb+i)
  lookback_s = lookback.strftime("%Y-%m")
  lookback_e = parse(lookback_s) + relativedelta(months=period_lb-1)

  train = returns.loc[lookback_s:lookback_e]
  estimate[i] = history(train)
  # space for optimization code used estimate
  backtest[i] = np.inner(returns.iloc[i], weights)
```

- (중간) 4회 반복에 대한 estimate 테이블이다.
- 반환된 Lookback period의 평균과 표준편차가 저장된다.

- 여기서는 최적화는 진행하지 않았지만 구해진 weights를 통해 투자 기간 동안의 수익률을 backtest table에 저장한다.




```
df(estimate, columns=['mean', 'std'])
```

	mean	std
0	0.020703	0.089028
1	0.019092	0.088587
2	0.009025	0.078981
3	0.012260	0.078423
4	0.000000	0.000000
5	0.000000	0.000000
6	0.000000	0.000000
7	0.000000	0.000000
8	0.000000	0.000000
9	0.000000	0.000000
10	0.000000	0.000000
11	0.000000	0.000000

```
df(backtest, columns=['returns'])
```

	returns
0	-0.00954
1	0.00057
2	0.04785
3	-0.03821
4	-0.01094
5	-0.06849
6	-0.09299
7	0.00849
8	-0.09801
9	0.06674
10	0.05964
11	-0.04435

Part 6 Demonstration

감사합니다