In [2]:
```
import pandas as pd
import numpy as np
```

In [229]:
```
#read in ffdata
ffdata = pd.read_csv("ffdata.txt", delim_whitespace=True)
ffdata
```

Out[229]:

	-0.000800	0.008300	0.004100	0.000040
0	0.0122	0.0035	0.0002	0.00004
1	0.0019	0.0012	0.0018	0.00004
2	0.0027	0.0050	-0.0007	0.00004
3	0.0048	0.0033	0.0062	0.00004
4	-0.0063	0.0003	0.0027	0.00004
...	...	...	...	...
1505	0.0052	-0.0008	0.0016	0.00000
1506	0.0010	-0.0015	-0.0030	0.00000
1507	-0.0009	0.0008	-0.0008	0.00000
1508	-0.0007	-0.0004	-0.0018	0.00000
1509	-0.0088	-0.0013	0.0015	0.00000

1510 rows × 4 columns

In [3]:
```
#load in ticker.txt - tickers
ticker = pd.read_csv('ticker.txt', header=None)
ticker = ticker.reset_index()
ticker['index'] = ticker['index'] + 1
ticker
```

Out[3]:

	index	0
0	1	A
1	2	AA
2	3	AAI
3	4	AAON
4	5	AAP
...	...	...
1872	1873	ZMH
1873	1874	ZOLL
1874	1875	ZOLT
1875	1876	ZQK
1876	1877	ZRAN

1877 rows × 2 columns

In [4]:
```
#load in retdate.txt - return dates
retdate = pd.read_csv('retdate.txt', sep = " ", header = None)
retdate[0]
```

Out[4]:
```
0       20040102
1       20040105
2       20040106
3       20040107
4       20040108
          ...
1506    20091224
1507    20091228
1508    20091229
1509    20091230
1510    20091231
Name: 0, Length: 1511, dtype: int64
```

In [5]:
```
#load in secdata.txt - securities data
secdata = pd.read_csv('secdata.txt', sep = " ", header = None)
secdata.columns = ['Ticker #', 'Stock Returns', 'Market Capitalizations'
]
secdata
```

Out[5]:

	Ticker #	Stock Returns	Market Capitalizations
0	1	-0.015048	13713091.20
1	1	0.026042	14070202.95
2	1	0.031472	14513021.52
3	1	0.012795	14698719.63
4	1	0.045675	15370089.72
...	...	...	...
2836142	1877	0.000000	580951.00
2836143	1877	-0.008079	576257.50
2836144	1877	0.000000	576257.50
2836145	1877	0.008145	580951.00
2836146	1877	-0.008079	576257.50

2836147 rows × 3 columns

In [6]:
```
#merge ticker with stocks
stock_with_ticker = secdata.merge(ticker, left_on = "Ticker #", right_on
= "index")
stock_with_ticker.to_csv('stock_with_ticker.csv')
```

Question 2

We now consider the modern portfolio theory (MPT) approach to estimating volatility. Each step below should be completed using 504, 252, 126, and 63 day rolling windows.

a) Pick a portfolio of 100 securities.

Criteria: Security numbers chosen at random between 1-1877.

In [7]:
```
secdata
```

Out[7]:

	Ticker #	Stock Returns	Market Capitalizations
0	1	-0.015048	13713091.20
1	1	0.026042	14070202.95
2	1	0.031472	14513021.52
3	1	0.012795	14698719.63
4	1	0.045675	15370089.72
...	...	...	...
2836142	1877	0.000000	580951.00
2836143	1877	-0.008079	576257.50
2836144	1877	0.000000	576257.50
2836145	1877	0.008145	580951.00
2836146	1877	-0.008079	576257.50

2836147 rows × 3 columns

In [8]:
```
random_tickers = list(np.random.choice(1511, 100))
```

In [9]:
```
#turn tickers into columns
secdata_group = secdata.set_index([secdata.groupby('Ticker #').cumcount
(), 'Ticker #'])['Stock Returns'].unstack()
secdata_group
```

Out[9]:

Ticker #	1	2	3	4	5	6	7	8	
0	-0.015048	-0.011842	0.030252	-0.024729	0.000246	-0.004212	-0.072416	-0.038290	0.0
1	0.026042	0.032756	0.008157	0.004754	-0.009089	0.041823	-0.072706	0.033889	0.0
2	0.031472	-0.007478	0.062298	0.014196	0.017848	-0.003608	0.097044	-0.004120	0.0
3	0.012795	-0.007534	0.022848	-0.005184	0.031417	0.022635	0.062683	0.009712	0.0
4	0.045675	0.012042	-0.067014	0.001042	-0.026446	0.034086	0.006064	-0.004809	0.0
...	...	...	...	...	...	...	...	...	
1506	0.000990	0.021250	0.009363	0.011190	0.000242	0.034339	0.019135	0.000760	0.0
1507	0.002308	-0.014688	-0.035250	-0.010060	-0.003867	0.012294	0.016327	0.002658	0.0
1508	-0.002303	-0.004348	-0.007692	0.013720	-0.011160	-0.011861	0.012450	-0.001515	0.0
1509	0.025387	0.016843	0.007752	-0.005013	0.006133	0.012147	0.015470	-0.002275	0.0
1510	-0.000965	-0.011043	0.003846	-0.018136	-0.012924	-0.004290	-0.001953	-0.009122	0.0

1511 rows × 1877 columns

In [10]:
```
portfolio_q2_ret = secdata_group[random_tickers]
portfolio_q2_ret
```

Out[10]:

Ticker #	861	895	747	1183	1389	495	729	656	
0	-0.008346	0.053533	-0.066890	-0.001489	0.019407	-0.015298	0.030151	-0.009652	0.(
1	0.008926	0.008638	-0.025090	-0.001491	0.063218	-0.000675	0.006098	-0.016243	-0.(
2	0.111476	0.017632	0.069853	0.008962	0.013514	0.076715	-0.030303	0.000144	-0.(
3	-0.044803	-0.000495	-0.020619	0.045522	0.024667	0.005650	-0.031250	0.000287	0.(
4	0.017619	0.054978	-0.070175	0.008850	0.119714	0.014045	0.032258	-0.008324	0.(
...	...	...	...	...	...	...	...	...	
1506	0.000855	0.008009	0.006702	0.007065	-0.000467	0.000204	-0.001992	0.002577	0.(
1507	-0.007689	0.013999	-0.029294	-0.004300	-0.002336	-0.002651	0.007984	0.006542	-0.(
1508	0.000000	0.001119	0.001372	-0.001818	0.011241	-0.004499	0.011881	0.005571	0.(
1509	-0.009901	0.011554	-0.023288	-0.009107	0.006484	0.000000	-0.025440	-0.000692	-0.(
1510	-0.010435	-0.017318	-0.018233	0.000460	-0.023470	-0.007806	0.014056	-0.002079	-0.(

1511 rows × 100 columns

In [11]:
```
#market cap grouped by ticker #
secdata_cap_group = secdata.set_index([secdata.groupby('Ticker #').cumco
unt(),
                                       'Ticker #'])['Market Capitalizati
ons'].unstack()
secdata_cap_group
```

Out[11]:

Ticker #	1	2	3	4	5	6	7	
0	13713091.20	32494080.25	1031397.02	237003.60	3004886.52	7.847234e+06	325464.88	6
1	14070202.95	33558466.90	1039809.72	238130.40	2977576.08	8.175431e+06	301801.76	6
2	14513021.52	33307513.95	1104587.51	241510.80	3030720.72	8.145930e+06	331089.72	6
3	14698719.63	33056561.00	1129825.61	240258.80	3125938.20	8.330311e+06	351843.44	6
4	15370089.72	33454624.30	1054111.31	240509.20	3043268.76	8.614257e+06	353977.00	6
...	...	...	...	...	...	...	...	
1506	10467246.96	15921336.52	725246.06	342293.84	3915830.78	1.882777e+08	539539.00	7
1507	10491404.80	15687485.80	699680.80	338850.24	3900689.82	1.905925e+08	548347.80	7
1508	10467246.96	15619279.34	694298.64	343499.10	3857159.56	1.883318e+08	555174.62	7
1509	10732983.20	15882361.40	699680.80	341777.30	3880817.31	1.906195e+08	563763.20	7
1510	10838179.17	15706973.36	702371.88	335578.82	3830662.88	1.898017e+08	562662.10	7

1511 rows × 1877 columns

In [47]:
```
portfolio_q2_cap = secdata_cap_group[random_tickers]
portfolio_q2_cap
```

Out[47]:

Ticker #	861	895	747	1183	1389	495	729	
0	3347083.23	7680552.96	79392.24	1786802.04	631299.84	3412256.40	182064.60	10283
1	3376960.28	7746899.20	77400.32	1784137.16	671209.60	3409951.60	183174.75	10116
2	3753411.11	7883494.40	82806.96	1800126.44	680280.00	3671546.40	177624.00	10117
3	3585246.00	7879591.68	81099.60	1882071.50	697060.24	3692289.60	172073.25	10120
4	3648414.62	8312793.60	75408.40	1898727.00	781919.14	3744147.60	177624.00	10036
...	...	...	...	...	...	...	...	
1506	4760540.55	13887352.77	386524.68	2734212.06	972309.00	4327485.86	123291.09	6434
1507	4723936.65	14081765.20	375201.72	2722456.00	970037.25	4316011.80	124275.45	6476
1508	4723936.65	14097528.37	375716.40	2717506.08	980941.65	4296594.16	125751.99	6512
1509	4677165.00	14260414.46	366966.84	2692756.48	987302.55	4296594.16	122552.82	6508
1510	4628359.80	14013458.13	360276.00	2693993.96	964130.70	4263054.60	127330.70	6494

1511 rows × 100 columns

For part b:

Standard deviation of portfolio = portfolio volatility.

Equation:

$$\hat{\sigma}_{Portfolio} = \sqrt{w_T \cdot \Sigma \cdot w}$$

where:

- w is portfolio weights
- Σ is covariance matrix
- $\cdot$ the dot-multiplication for matrix multiplication
- $\hat{\sigma}_{Portfolio}$ is the estimated portfolio volatility/standard deviation

In [178]:
```
#function to find portfolio standard deviation
def sd_portfolio(cov_mat, arr_weights):
    if np.isnan(cov_mat).any():
        return cov_mat
    return np.dot(np.dot(np.transpose(arr_weights), cov_mat), arr_weight
s)**0.5
```

Rolling Window 504

i) Generate a covariance matrixes for generated portfolio.

In [156]:
```
cov_df_504 = portfolio_q2_ret.rolling(504).cov()
cov_df_504.dropna(inplace = True)
cov_df_504.drop(cov_df_504.tail(100).index, inplace = True)
cov_df_504_np = cov_df_504.to_numpy()
cov_df_504
```

Out[156]:

	Ticker #	861	895	747	1183	1389	495	729	656
	Ticker #								
503	**861**	0.000253	0.000047	0.000010	0.000010	0.000056	0.000052	0.000031	0.000024
	895	0.000047	0.000758	-0.000015	0.000162	0.000178	0.000127	0.000121	0.000036
	747	0.000010	-0.000015	0.001621	0.000120	0.000018	0.000070	0.000138	0.000092
	1183	0.000010	0.000162	0.000120	0.000734	0.000149	0.000118	0.000112	0.000080
	1389	0.000056	0.000178	0.000018	0.000149	0.000690	0.000108	0.000074	0.000051
...	**...**	...	...	...	...	...	...	...	...
1509	**1196**	0.000435	0.000503	0.000801	0.000385	0.000679	0.000321	0.000851	0.000542
	163	0.000481	0.000668	0.000702	0.000412	0.000574	0.000281	0.000877	0.000564
	388	0.000594	0.000669	0.001035	0.000546	0.000719	0.000330	0.001112	0.000697
	610	0.000865	0.001076	0.001356	0.000896	0.001059	0.000442	0.001989	0.001171
	304	0.000638	0.000808	0.000966	0.000513	0.000860	0.000368	0.001297	0.000715

100700 rows × 100 columns

ii) Estimate the standard deviations of the portfolio over rw (from the last day in the rolling window).

In [186]:
```
#portfolio_weights
weights_q2_504 = portfolio_q2_cap.iloc[:, :].apply(lambda x: x.div(x.sum
()), axis=1)
df_weights_q2_504 = weights_q2_504.drop(weights_q2_504.tail(1).index)
arr_weights_q2_504 = df_weights_q2_504.tail(1007).to_numpy()
df_weights_q2_504
```

Out[186]:

Ticker #	861	895	747	1183	1389	495	729	656	1479
0	0.004805	0.011026	0.000114	0.002565	0.000906	0.004899	0.000261	0.014763	0.002665
1	0.004812	0.011038	0.000110	0.002542	0.000956	0.004859	0.000261	0.014413	0.002644
2	0.005343	0.011223	0.000118	0.002563	0.000968	0.005227	0.000253	0.014403	0.002635
3	0.005049	0.011096	0.000114	0.002650	0.000982	0.005199	0.000242	0.014251	0.002623
4	0.005130	0.011688	0.000106	0.002670	0.001099	0.005264	0.000250	0.014111	0.002622
...	...	...	...	...	...	...	...	...	...
1505	0.007734	0.022400	0.000624	0.004414	0.001582	0.007035	0.000201	0.010435	0.003216
1506	0.007695	0.022449	0.000625	0.004420	0.001572	0.006995	0.000199	0.010402	0.003235
1507	0.007622	0.022722	0.000605	0.004393	0.001565	0.006964	0.000201	0.010451	0.003208
1508	0.007640	0.022801	0.000608	0.004395	0.001587	0.006949	0.000203	0.010533	0.003254
1509	0.007562	0.023055	0.000593	0.004353	0.001596	0.006946	0.000198	0.010522	0.003245

1510 rows × 100 columns

In [206]:
```
sd_port_504 = []
for i in range(0, len(cov_df_504_np), 100):
    idx = 0
    sd_port_504.append(sd_portfolio(cov_df_504_np[i:i+100], arr_weights_
q2_504[idx]))
    idx += 1
sd_port_504_arr = np.array(sd_port_504)
sd_port_504_arr
```

Out[206]:
```
array([0.00778539, 0.0078214 , 0.00781648, ..., 0.03085613, 0.03085523,
       0.03085458])
```

iii) Using the market capitalization weights and returns (from the day following the last day in the rolling window) of your securities, calculate the one-day ahead return of the portfolio, $\tilde{r}_p$.

In [230]:
```
#getting one-day ahead returns array
dayahead504_port_ret_q2 = []
dayahead504_ret_q2 = portfolio_q2_ret.loc[504:1510].to_numpy()
dayahead504_w_q2 = weights_q2_504.loc[504:1510].to_numpy()
for i in range(0, len(dayahead504_w_q2)):
    dayahead504_port_ret_q2.append(np.multiply(dayahead504_ret_q2[i], da
yahead504_w_q2[i]))
dayahead504_port_ret_q2_arr = np.sum(np.array(dayahead504_port_ret_q2),
axis = 1)
dayahead504_port_ret_q2_arr
```

Out[230]:
```
array([ 0.01755353,  0.00136112,  0.00178823, ..., -0.0022362 ,
        0.0004401 , -0.00969607])
```

iv) Calculate the standardized outcome, $\tilde{z}_p$, where $\tilde{z}_p = \frac{\tilde{r}_p}{\hat{\sigma}_p}$ where we make the simplifying assumption that $E[\tilde{r}_p] = 0$.

In [231]:
```
#dividing arrays to get standardized outcomes.
standardized_outcomes_504_q2 = dayahead504_port_ret_q2_arr / sd_port_504
_arr
std_outcomes_504_q2 = pd.DataFrame(standardized_outcomes_504_q2)
std_outcomes_504_q2.index += 504
std_outcomes_504_q2.rename(columns={0: "Standardized Outcome"}, inplace
= True)
std_outcomes_504_q2
```

Out[231]:

	Standardized Outcome
504	2.254676
505	0.174026
506	0.228777
507	0.402387
508	0.603898
...	...
1506	0.190636
1507	0.060392
1508	-0.072472
1509	0.014263
1510	-0.314250

1007 rows × 1 columns

Rolling Window 252

i) Generate a covariance matrixes for generated portfolio.

In [219]:
```
cov_df_252 = portfolio_q2_ret.rolling(252).cov()
cov_df_252.dropna(inplace = True)
cov_df_252.drop(cov_df_252.tail(100).index, inplace = True)
cov_df_252_np = cov_df_252.to_numpy()
cov_df_252
```

Out[219]:

	Ticker #	861	895	747	1183	1389	495	729	6
	Ticker #								
251	**861**	2.308780e-04	0.000064	0.000024	-6.332419e-07	0.000084	0.000063	0.000048	0.0000
	895	6.351895e-05	0.001165	-0.000017	2.569938e-04	0.000286	0.000179	0.000217	0.0000
	747	2.428240e-05	-0.000017	0.001943	1.345425e-04	0.000020	0.000077	0.000240	0.0001
	1183	-6.332419e-07	0.000257	0.000135	1.044115e-03	0.000211	0.000171	0.000193	0.0000
	1389	8.371464e-05	0.000286	0.000020	2.111205e-04	0.000800	0.000173	0.000167	0.0000
...	**...**	...	...	...	...	...	...	...	
1509	**1196**	3.047874e-04	0.000373	0.000809	3.492820e-04	0.000479	0.000042	0.000931	0.0005
	163	3.736479e-04	0.000512	0.000679	3.559447e-04	0.000444	0.000074	0.000919	0.0005
	388	4.292569e-04	0.000721	0.001157	6.273019e-04	0.000618	0.000081	0.001152	0.0007
	610	8.807421e-04	0.001093	0.001623	1.088934e-03	0.001048	0.000133	0.002666	0.0014
	304	4.811334e-04	0.000603	0.000867	4.658249e-04	0.000644	0.000077	0.001411	0.0006

125900 rows × 100 columns

ii) Estimate the standard deviations of the portfolio over rw (from the last day in the rolling window).

In [223]:
```
#portfolio_weights
weights_q2_252 = portfolio_q2_cap.iloc[:, :].apply(lambda x: x.div(x.sum
()), axis=1)
df_weights_q2_252 = weights_q2_252.drop(weights_q2_252.tail(1).index)
arr_weights_q2_252 = df_weights_q2_252.tail(1259).to_numpy()
df_weights_q2_252
```

Out[223]:

Ticker #	861	895	747	1183	1389	495	729	656	1479
0	0.004805	0.011026	0.000114	0.002565	0.000906	0.004899	0.000261	0.014763	0.002665
1	0.004812	0.011038	0.000110	0.002542	0.000956	0.004859	0.000261	0.014413	0.002644
2	0.005343	0.011223	0.000118	0.002563	0.000968	0.005227	0.000253	0.014403	0.002635
3	0.005049	0.011096	0.000114	0.002650	0.000982	0.005199	0.000242	0.014251	0.002623
4	0.005130	0.011688	0.000106	0.002670	0.001099	0.005264	0.000250	0.014111	0.002622
...	...	...	...	...	...	...	...	...	...
1505	0.007734	0.022400	0.000624	0.004414	0.001582	0.007035	0.000201	0.010435	0.003216
1506	0.007695	0.022449	0.000625	0.004420	0.001572	0.006995	0.000199	0.010402	0.003235
1507	0.007622	0.022722	0.000605	0.004393	0.001565	0.006964	0.000201	0.010451	0.003208
1508	0.007640	0.022801	0.000608	0.004395	0.001587	0.006949	0.000203	0.010533	0.003254
1509	0.007562	0.023055	0.000593	0.004353	0.001596	0.006946	0.000198	0.010522	0.003245

1510 rows × 100 columns

In [228]:
```
sd_port_252 = []
for i in range(0, len(cov_df_252_np), 100):
    idx = 0
    sd_port_252.append(sd_portfolio(cov_df_252_np[i:i+100], arr_weights_
q2_252[idx]))
    idx += 1
sd_port_252_arr = np.array(sd_port_252)
sd_port_252_arr
```

Out[228]:
```
array([0.00836495, 0.00839972, 0.00846015, ..., 0.03076475, 0.03075179,
       0.03070573])
```

iii) Using the market capitalization weights and returns (from the day following the last day in the rolling window) of your securities, calculate the one-day ahead return of the portfolio, $\tilde{r}_p$.

In [239]:
```
#getting one-day ahead returns array
dayahead252_port_ret_q2 = []
dayahead252_ret_q2 = portfolio_q2_ret.loc[252:1510].to_numpy()
dayahead252_w_q2 = weights_q2_252.loc[252:1510].to_numpy()
for i in range(0, len(dayahead252_w_q2)):
    dayahead252_port_ret_q2.append(np.multiply(dayahead252_ret_q2[i], da
yahead252_w_q2[i]))
dayahead252_port_ret_q2_arr = np.sum(np.array(dayahead252_port_ret_q2),
axis = 1)
dayahead252_port_ret_q2_arr
```

Out[239]:
```
array([-0.01187761, -0.01687497, -0.00844107, ..., -0.0022362 ,
        0.0004401 , -0.00969607])
```

iv) Calculate the standardized outcome, $\tilde{z}_p$, where $\tilde{z}_p = \frac{\tilde{r}_p}{\hat{\sigma}_p}$ where we make the simplifying assumption that $E[\tilde{r}_p] = 0$.

In [235]:
```
standardized_outcomes_252_q2 = dayahead252_port_ret_q2_arr / sd_port_252
_arr
std_outcomes_252_q2 = pd.DataFrame(standardized_outcomes_252_q2)
std_outcomes_252_q2.index += 252
std_outcomes_252_q2.rename(columns={0: "Standardized Outcome"}, inplace
= True)
std_outcomes_252_q2
```

Out[235]:

	Standardized Outcome
252	-1.419925
253	-2.008991
254	-0.997745
255	0.437783
256	-0.201522
...	...
1506	0.191126
1507	0.060589
1508	-0.072687
1509	0.014311
1510	-0.315774

1259 rows × 1 columns

Rolling Window 126

i) Generate a covariance matrixes for generated portfolio.

In [236]:
```
cov_df_126 = portfolio_q2_ret.rolling(126).cov()
cov_df_126.dropna(inplace = True)
cov_df_126.drop(cov_df_126.tail(100).index, inplace = True)
cov_df_126_np = cov_df_126.to_numpy()
cov_df_126
```

Out[236]:

	Ticker #	861	895	747	1183	1389	495	729	656
	Ticker #								
125	861	0.000240	0.000083	0.000066	0.000011	0.000073	0.000091	0.000065	0.000028
	895	0.000083	0.001739	-0.000155	0.000373	0.000368	0.000155	0.000307	0.000053
	747	0.000066	-0.000155	0.003010	0.000171	-0.000021	0.000128	0.000372	0.000168
	1183	0.000011	0.000373	0.000171	0.001161	0.000284	0.000173	0.000251	0.000110
	1389	0.000073	0.000368	-0.000021	0.000284	0.000833	0.000122	0.000212	0.000061
...	...	...	...	...	...	...	...	...	...
1509	1196	0.000077	0.000157	0.000190	0.000138	0.000150	0.000030	0.000175	0.000195
	163	0.000126	0.000253	0.000230	0.000194	0.000128	0.000107	0.000269	0.000238
	388	0.000171	0.000285	0.000430	0.000215	0.000225	0.000048	0.000431	0.000287
	610	0.000415	0.000271	0.000545	0.000345	0.000525	0.000172	0.000491	0.000558
	304	0.000184	0.000287	0.000274	0.000210	0.000265	0.000057	0.000413	0.000260

138500 rows × 100 columns

ii) Estimate the standard deviations of the portfolio over rw (from the last day in the rolling window).

In [237]:
```
#portfolio_weights
weights_q2_126 = portfolio_q2_cap.iloc[:, :].apply(lambda x: x.div(x.sum
()), axis=1)
df_weights_q2_126 = weights_q2_126.drop(weights_q2_126.tail(1).index)
arr_weights_q2_126 = df_weights_q2_126.tail(1259).to_numpy()
df_weights_q2_126
```

Out[237]:

Ticker #	861	895	747	1183	1389	495	729	656	1479
0	0.004805	0.011026	0.000114	0.002565	0.000906	0.004899	0.000261	0.014763	0.002665
1	0.004812	0.011038	0.000110	0.002542	0.000956	0.004859	0.000261	0.014413	0.002644
2	0.005343	0.011223	0.000118	0.002563	0.000968	0.005227	0.000253	0.014403	0.002635
3	0.005049	0.011096	0.000114	0.002650	0.000982	0.005199	0.000242	0.014251	0.002623
4	0.005130	0.011688	0.000106	0.002670	0.001099	0.005264	0.000250	0.014111	0.002622
...	...	...	...	...	...	...	...	...	...
1505	0.007734	0.022400	0.000624	0.004414	0.001582	0.007035	0.000201	0.010435	0.003216
1506	0.007695	0.022449	0.000625	0.004420	0.001572	0.006995	0.000199	0.010402	0.003235
1507	0.007622	0.022722	0.000605	0.004393	0.001565	0.006964	0.000201	0.010451	0.003208
1508	0.007640	0.022801	0.000608	0.004395	0.001587	0.006949	0.000203	0.010533	0.003254
1509	0.007562	0.023055	0.000593	0.004353	0.001596	0.006946	0.000198	0.010522	0.003245

1510 rows × 100 columns

In [238]:
```
sd_port_126 = []
for i in range(0, len(cov_df_126_np), 100):
    idx = 0
    sd_port_126.append(sd_portfolio(cov_df_126_np[i:i+100], arr_weights_
q2_126[idx]))
    idx += 1
sd_port_126_arr = np.array(sd_port_126)
sd_port_126_arr
```

Out[238]:
```
array([0.00827518, 0.0083208 , 0.00830922, ..., 0.01361003, 0.01361296,
       0.01360251])
```

iii) Using the market capitalization weights and returns (from the day following the last day in the rolling window) of your securities, calculate the one-day ahead return of the portfolio, $\tilde{r}_p$.

In [240]:
```
#getting one-day ahead returns array
dayahead126_port_ret_q2 = []
dayahead126_ret_q2 = portfolio_q2_ret.loc[126:1510].to_numpy()
dayahead126_w_q2 = weights_q2_126.loc[126:1510].to_numpy()
for i in range(0, len(dayahead126_w_q2)):
    dayahead126_port_ret_q2.append(np.multiply(dayahead126_ret_q2[i], da
yahead126_w_q2[i]))
dayahead126_port_ret_q2_arr = np.sum(np.array(dayahead126_port_ret_q2),
axis = 1)
dayahead126_port_ret_q2_arr
```

Out[240]:
```
array([-0.00986861,  0.0042113 , -0.00609825, ..., -0.0022362 ,
        0.0004401 , -0.00969607])
```

iv) Calculate the standardized outcome, $\tilde{z}_p$, where $\tilde{z}_p = \frac{\tilde{r}_p}{\hat{\sigma}_p}$ where we make the simplifying assumption that $E[\tilde{r}_p] = 0$.

In [241]:
```
standardized_outcomes_126_q2 = dayahead126_port_ret_q2_arr / sd_port_126
_arr
std_outcomes_126_q2 = pd.DataFrame(standardized_outcomes_126_q2)
std_outcomes_126_q2.index += 126
std_outcomes_126_q2.rename(columns={0: "Standardized Outcome"}, inplace
= True)
std_outcomes_126_q2
```

Out[241]:

	Standardized Outcome
126	-1.192556
127	0.506118
128	-0.733914
129	0.621606
130	0.203769
...	...
1506	0.430622
1507	0.136569
1508	-0.164305
1509	0.032330
1510	-0.712815

1385 rows × 1 columns

Rolling Window 63

i) Generate a covariance matrixes for generated portfolio.

In [242]:
```
cov_df_63 = portfolio_q2_ret.rolling(63).cov()
cov_df_63.dropna(inplace = True)
cov_df_63.drop(cov_df_63.tail(100).index, inplace = True)
cov_df_63_np = cov_df_63.to_numpy()
cov_df_63
```

Out[242]:

	Ticker #	861	895	747	1183	1389	495	729	656
	Ticker #								
62	861	0.000378	0.000110	0.000031	0.000029	0.000090	0.000160	0.000043	0.000020
	895	0.000110	0.002420	-0.000420	0.000287	0.000366	0.000087	0.000265	0.000002
	747	0.000031	-0.000420	0.004627	0.000423	-0.000177	0.000232	0.000471	0.000208
	1183	0.000029	0.000287	0.000423	0.000756	0.000292	0.000121	0.000206	0.000104
	1389	0.000090	0.000366	-0.000177	0.000292	0.000974	0.000060	0.000209	-0.000007
...	...	...	...	...	...	...	...	...	...
1509	1196	0.000084	0.000091	0.000175	0.000088	0.000100	0.000039	0.000187	0.000114
	163	0.000062	0.000265	0.000315	0.000237	0.000156	0.000091	0.000263	0.000252
	388	0.000115	0.000231	0.000527	0.000249	0.000210	0.000087	0.000315	0.000282
	610	0.000471	0.000310	0.000907	0.000459	0.000703	0.000171	0.000747	0.000624
	304	0.000213	0.000302	0.000377	0.000244	0.000243	0.000124	0.000374	0.000324

144800 rows × 100 columns

ii) Estimate the standard deviations of the portfolio over rw (from the last day in the rolling window).

In [243]:
```
#portfolio_weights
weights_q2_63 = portfolio_q2_cap.iloc[:, :].apply(lambda x: x.div(x.sum
()), axis=1)
df_weights_q2_63 = weights_q2_63.drop(weights_q2_63.tail(1).index)
arr_weights_q2_63 = df_weights_q2_63.tail(1448).to_numpy()
df_weights_q2_63
```

Out[243]:

Ticker #	861	895	747	1183	1389	495	729	656	1479
0	0.004805	0.011026	0.000114	0.002565	0.000906	0.004899	0.000261	0.014763	0.002665
1	0.004812	0.011038	0.000110	0.002542	0.000956	0.004859	0.000261	0.014413	0.002644
2	0.005343	0.011223	0.000118	0.002563	0.000968	0.005227	0.000253	0.014403	0.002635
3	0.005049	0.011096	0.000114	0.002650	0.000982	0.005199	0.000242	0.014251	0.002623
4	0.005130	0.011688	0.000106	0.002670	0.001099	0.005264	0.000250	0.014111	0.002622
...	...	...	...	...	...	...	...	...	...
1505	0.007734	0.022400	0.000624	0.004414	0.001582	0.007035	0.000201	0.010435	0.003216
1506	0.007695	0.022449	0.000625	0.004420	0.001572	0.006995	0.000199	0.010402	0.003235
1507	0.007622	0.022722	0.000605	0.004393	0.001565	0.006964	0.000201	0.010451	0.003208
1508	0.007640	0.022801	0.000608	0.004395	0.001587	0.006949	0.000203	0.010533	0.003254
1509	0.007562	0.023055	0.000593	0.004353	0.001596	0.006946	0.000198	0.010522	0.003245

1510 rows × 100 columns

In [248]:
```
sd_port_63 = []
for i in range(0, len(cov_df_63_np), 100):
    idx = 0
    sd_port_63.append(sd_portfolio(cov_df_63_np[i:i+100], arr_weights_q2
_63[idx]))
    idx += 1
sd_port_63_arr = np.array(sd_port_63)
sd_port_63_arr
```

Out[248]:
```
array([0.00910965, 0.0092089 , 0.0092066 , ..., 0.01296528, 0.01296553,
       0.01294103])
```

iii) Using the market capitalization weights and returns (from the day following the last day in the rolling window) of your securities, calculate the one-day ahead return of the portfolio, $\tilde{r}_p$.

In [246]:
```
#getting one-day ahead returns array
dayahead63_port_ret_q2 = []
dayahead63_ret_q2 = portfolio_q2_ret.loc[63:1510].to_numpy()
dayahead63_w_q2 = weights_q2_63.loc[63:1510].to_numpy()
for i in range(0, len(dayahead63_w_q2)):
    dayahead63_port_ret_q2.append(np.multiply(dayahead63_ret_q2[i], daya
head63_w_q2[i]))
dayahead63_port_ret_q2_arr = np.sum(np.array(dayahead63_port_ret_q2), ax
is = 1)
dayahead63_port_ret_q2_arr
```

Out[246]:
```
array([ 0.01076997,  0.00706012, -0.00490281, ..., -0.0022362 ,
        0.0004401 , -0.00969607])
```

iv) Calculate the standardized outcome, $\tilde{z}_p$, where $\tilde{z}_p = \frac{\tilde{r}_p}{\hat{\sigma}_p}$ where we make the simplifying assumption that $E[\tilde{r}_p] = 0$.

In [249]:
```
standardized_outcomes_63_q2 = dayahead63_port_ret_q2_arr / sd_port_63_ar
r
std_outcomes_63_q2 = pd.DataFrame(standardized_outcomes_63_q2)
std_outcomes_63_q2.index += 63
std_outcomes_63_q2.rename(columns={0: "Standardized Outcome"}, inplace =
True)
std_outcomes_63_q2
```

Out[249]:

	Standardized Outcome
63	1.182259
64	0.766663
65	-0.532532
66	-0.840335
67	-0.587595
...	...
1506	0.444428
1507	0.140960
1508	-0.172476
1509	0.033944
1510	-0.749250

1448 rows × 1 columns

b) Compute bias statistics.

In [250]:
```
bias_stat_q2_504 = np.std(standardized_outcomes_504_q2)
bias_stat_q2_252 = np.std(standardized_outcomes_252_q2)
bias_stat_q2_126 = np.std(standardized_outcomes_126_q2)
bias_stat_q2_63 = np.std(standardized_outcomes_63_q2)
print(bias_stat_q2_504, bias_stat_q2_252, bias_stat_q2_126, bias_stat_q2
_63)
```

```
1.3204917779811967 1.0903814842318365 1.018551730690755 0.9926056945556
908
```

Rolling window 63 gives closest bias statistic to 1. Idk why.