In [54]:
```
import pandas as pd
import numpy as np
from statsmodels.regression.rolling import RollingOLS
from scipy import stats
import statsmodels.regression.linear_model as sm
import statsmodels.tools.tools as sm2
from statsmodels.regression.linear_model import OLS
#from pyfinance import PandasRollingOLS
from statsmodels.api import add_constant
```

In [308]:
```
#read in ffdata
new_row = pd.DataFrame({'-0.000800':-0.000800, '0.008300':0.008300, '0.0
04100':0.004100,
                        '0.000040':0.000040}, index = [0])
ffdata = pd.read_csv("ffdata.txt", delim_whitespace=True)
ffdata = pd.concat([new_row, ffdata]).reset_index(drop = True)
ffdata.rename(columns={'-0.000800':'Market Returns', '0.008300':'Returns
to the Fama-French size factor',
                       '0.004100':'Returns to the Fama-French value fact
or',
                       '0.000040':'Risk-free rate'}, inplace = True)
ffdata
```

Out[308]:

	Market Returns	Returns to the Fama-French size factor	Returns to the Fama-French value factor	Risk-free rate
0	-0.0008	0.0083	0.0041	0.00004
1	0.0122	0.0035	0.0002	0.00004
2	0.0019	0.0012	0.0018	0.00004
3	0.0027	0.0050	-0.0007	0.00004
4	0.0048	0.0033	0.0062	0.00004
...	...	...	...	...
1506	0.0052	-0.0008	0.0016	0.00000
1507	0.0010	-0.0015	-0.0030	0.00000
1508	-0.0009	0.0008	-0.0008	0.00000
1509	-0.0007	-0.0004	-0.0018	0.00000
1510	-0.0088	-0.0013	0.0015	0.00000

1511 rows × 4 columns

In [296]:
```
#load in ticker.txt - tickers
ticker = pd.read_csv('ticker.txt', header=None)
ticker = ticker.reset_index()
ticker['index'] = ticker['index'] + 1
ticker.head()
```

Out[296]:

	index	0
0	1	A
1	2	AA
2	3	AAI
3	4	AAON
4	5	AAP

In [204]:
```
#load in retdate.txt - return dates
retdate = pd.read_csv('retdate.txt', sep = " ", header = None)
retdate.loc[1510]
```

Out[204]:
```
0    20091231
Name: 1510, dtype: int64
```

In [297]:
```
#load in secdata.txt - securities data
secdata = pd.read_csv('secdata.txt', sep = " ", header = None)
secdata.columns = ['Ticker #', 'Stock Returns', 'Market Capitalizations'
]
secdata.head()
```

Out[297]:

	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

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [307]:
```
#merge ticker with stocks
stock_with_ticker = secdata.merge(ticker, left_on = "Ticker #", right_on
= "index")
stock_with_ticker
```

Out[307]:

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

2836147 rows × 5 columns

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: 1st 100 securities.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [60]:
```
secdata
```

Out[60]:

	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 [61]:
```
random_tickers = list(range(1,101))
```

In [62]:
```
#turn tickers into columns
secdata_group = secdata.set_index([secdata.groupby('Ticker #').cumcount
(), 'Ticker #'])['Stock Returns'].unstack()
secdata_group
```

Out[62]:

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 [63]:
```
portfolio_q2_ret = secdata_group[random_tickers]
portfolio_q2_ret
```

Out[63]:

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 × 100 columns

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [64]:

```
#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[64]:

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 [65]:

```
portfolio_q2_cap = secdata_cap_group[random_tickers]
portfolio_q2_cap
```

Out[65]:

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 × 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 [66]:
```
#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.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [67]:
```
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[67]:

	Ticker #	1	2	3	4	5	6	7	8	
	Ticker #									
503	**1**	0.000438	0.000085	0.000203	0.000117	0.000071	0.000140	0.000142	0.000058	0.
	2	0.000085	0.000255	0.000123	0.000068	0.000062	0.000122	0.000086	0.000053	0.
	3	0.000203	0.000123	0.000910	0.000135	0.000131	0.000171	0.000280	0.000100	0.
	4	0.000117	0.000068	0.000135	0.000482	0.000065	0.000090	0.000122	0.000040	0.
	5	0.000071	0.000062	0.000131	0.000065	0.000316	0.000070	0.000099	0.000007	0.
...	**...**	...	...	...	...	...	...	...	...	
1509	**96**	0.000409	0.000568	0.000574	0.000460	0.000312	0.000247	0.000448	0.000218	0.
	97	0.000689	0.001155	0.001032	0.000897	0.000628	0.000641	0.000685	0.000323	0.
	98	0.000728	0.000893	0.002917	0.001070	0.000794	0.000664	0.000785	0.000371	0.
	99	0.000591	0.000803	0.000918	0.000784	0.000431	0.000454	0.000610	0.000323	0.
	100	0.000938	0.001578	0.001011	0.000973	0.000742	0.000784	0.000960	0.000419	0.

100700 rows × 100 columns

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

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [68]:
```
#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[68]:

Ticker #	1	2	3	4	5	6	7	8	9
0	0.018568	0.043998	0.001397	0.000321	0.004069	0.010625	0.000441	0.008199	0.00021
1	0.018809	0.044861	0.001390	0.000318	0.003980	0.010929	0.000403	0.008369	0.000217
2	0.019375	0.044465	0.001475	0.000322	0.004046	0.010875	0.000442	0.008323	0.00021
3	0.019478	0.043805	0.001497	0.000318	0.004142	0.011039	0.000466	0.008342	0.00021
4	0.020277	0.044135	0.001391	0.000317	0.004015	0.011364	0.000467	0.008265	0.00021
...	...	...	...	...	...	...	...	...	...
1505	0.013604	0.020282	0.000935	0.000440	0.005093	0.236809	0.000689	0.009864	0.00012
1506	0.013457	0.020469	0.000932	0.000440	0.005034	0.242061	0.000694	0.009756	0.00011
1507	0.013420	0.020067	0.000895	0.000433	0.004990	0.243796	0.000701	0.009732	0.00012
1508	0.013447	0.020065	0.000892	0.000441	0.004955	0.241941	0.000713	0.009759	0.00012
1509	0.013719	0.020300	0.000894	0.000437	0.004960	0.243645	0.000721	0.009688	0.00012

1510 rows × 100 columns

In [69]:
```
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[69]:
```
array([0.00820984, 0.00824556, 0.00823351, ..., 0.03161844, 0.03161853,
       0.03161824])
```

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, $\tilde{r}_p$.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [70]:
```
#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[70]:
```
array([ 0.01826034,  0.00564215, -0.00055066, ..., -0.00414611,
        0.00514476, -0.00946444])
```

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 [299]:
```
#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.head()
```

Out[299]:

	Standardized Outcome
504	2.224203
505	0.684265
506	-0.066881
507	1.336911
508	0.451453

Rolling Window 252

i) Generate a covariance matrixes for generated portfolio.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [72]:
```
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[72]:

	Ticker #	1	2	3	4	5	6	7	8	
	Ticker #									
251	**1**	0.000583	0.000118	0.000287	0.000161	0.000108	0.000154	0.000203	0.000078	0.
	2	0.000118	0.000327	0.000146	0.000086	0.000071	0.000127	0.000116	0.000072	0.
	3	0.000287	0.000146	0.000980	0.000183	0.000155	0.000187	0.000392	0.000126	0.
	4	0.000161	0.000086	0.000183	0.000538	0.000077	0.000067	0.000177	0.000052	0.
	5	0.000108	0.000071	0.000155	0.000077	0.000305	0.000057	0.000108	0.000014	0.
...	**...**	...	...	...	...	...	...	...	...	
1509	**96**	0.000361	0.000421	0.000348	0.000372	0.000114	0.000126	0.000305	0.000147	0.
	97	0.000467	0.000730	0.000631	0.000512	0.000308	0.000345	0.000420	0.000180	0.
	98	0.000628	0.001012	0.001642	0.000609	0.000306	0.000381	0.000467	0.000249	0.
	99	0.000455	0.000677	0.000675	0.000526	0.000225	0.000251	0.000378	0.000226	0.
	100	0.000725	0.001194	0.000937	0.000610	0.000317	0.000482	0.000615	0.000217	0.

125900 rows × 100 columns

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

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [73]:
```
#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[73]:

Ticker #	1	2	3	4	5	6	7	8	9
0	0.018568	0.043998	0.001397	0.000321	0.004069	0.010625	0.000441	0.008199	0.00021
1	0.018809	0.044861	0.001390	0.000318	0.003980	0.010929	0.000403	0.008369	0.000217
2	0.019375	0.044465	0.001475	0.000322	0.004046	0.010875	0.000442	0.008323	0.00021
3	0.019478	0.043805	0.001497	0.000318	0.004142	0.011039	0.000466	0.008342	0.00021
4	0.020277	0.044135	0.001391	0.000317	0.004015	0.011364	0.000467	0.008265	0.00021
...	...	...	...	...	...	...	...	...	...
1505	0.013604	0.020282	0.000935	0.000440	0.005093	0.236809	0.000689	0.009864	0.00012
1506	0.013457	0.020469	0.000932	0.000440	0.005034	0.242061	0.000694	0.009756	0.00011
1507	0.013420	0.020067	0.000895	0.000433	0.004990	0.243796	0.000701	0.009732	0.00012
1508	0.013447	0.020065	0.000892	0.000441	0.004955	0.241941	0.000713	0.009759	0.00012
1509	0.013719	0.020300	0.000894	0.000437	0.004960	0.243645	0.000721	0.009688	0.00012

1510 rows × 100 columns

In [74]:
```
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[74]:
```
array([0.00848636, 0.00849811, 0.00850478, ..., 0.03044609, 0.03043997,
       0.03041131])
```

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, $\tilde{r}_p$.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [75]:
```
#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[75]:
```
array([-0.00643605, -0.01146078, -0.00250408, ..., -0.00414611,
        0.00514476, -0.00946444])
```

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 [76]:
```
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[76]:

	Standardized Outcome
252	-0.758400
253	-1.348626
254	-0.294432
255	0.541728
256	0.640736
...	...
1506	0.397123
1507	0.170497
1508	-0.136179
1509	0.169013
1510	-0.311215

1259 rows × 1 columns

Rolling Window 126

Loading [MathJax]/jax/output/HTML-CSS/jax.js

i) Generate a covariance matrixes for generated portfolio.

In [77]:
```
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[77]:

	Ticker #	1	2	3	4	5	6	7	8	
	Ticker #									
125	**1**	0.000528	0.000192	0.000244	0.000163	0.000079	0.000183	0.000176	0.000096	0.
	2	0.000192	0.000408	0.000228	0.000095	0.000062	0.000162	0.000170	0.000098	0.
	3	0.000244	0.000228	0.001045	0.000181	0.000174	0.000204	0.000361	0.000105	0.
	4	0.000163	0.000095	0.000181	0.000595	0.000054	0.000046	0.000164	0.000057	0.
	5	0.000079	0.000062	0.000174	0.000054	0.000228	0.000089	0.000104	0.000034	0.
...	**...**	...	...	...	...	...	...	...	...	
1509	**96**	0.000177	0.000201	0.000162	0.000103	0.000020	0.000089	0.000153	0.000083	0.
	97	0.000233	0.000360	0.000289	0.000181	0.000154	0.000157	0.000230	0.000081	0.
	98	0.000241	0.000462	0.000875	0.000252	0.000170	0.000176	0.000315	0.000113	0.
	99	0.000180	0.000191	0.000192	0.000191	0.000055	0.000075	0.000078	0.000078	0.
	100	0.000318	0.000584	0.000173	0.000194	0.000197	0.000283	0.000283	0.000072	0.

138500 rows × 100 columns

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

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [78]:
```
#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(1385).to_numpy()
df_weights_q2_126
```

Out[78]:

Ticker #	1	2	3	4	5	6	7	8	9
0	0.018568	0.043998	0.001397	0.000321	0.004069	0.010625	0.000441	0.008199	0.000216
1	0.018809	0.044861	0.001390	0.000318	0.003980	0.010929	0.000403	0.008369	0.000217
2	0.019375	0.044465	0.001475	0.000322	0.004046	0.010875	0.000442	0.008323	0.000218
3	0.019478	0.043805	0.001497	0.000318	0.004142	0.011039	0.000466	0.008342	0.000219
4	0.020277	0.044135	0.001391	0.000317	0.004015	0.011364	0.000467	0.008265	0.000219
...	...	...	...	...	...	...	...	...	...
1505	0.013604	0.020282	0.000935	0.000440	0.005093	0.236809	0.000689	0.009864	0.000120
1506	0.013457	0.020469	0.000932	0.000440	0.005034	0.242061	0.000694	0.009756	0.000119
1507	0.013420	0.020067	0.000895	0.000433	0.004990	0.243796	0.000701	0.009732	0.000123
1508	0.013447	0.020065	0.000892	0.000441	0.004955	0.241941	0.000713	0.009759	0.000125
1509	0.013719	0.020300	0.000894	0.000437	0.004960	0.243645	0.000721	0.009688	0.000126

1510 rows × 100 columns

In [79]:
```
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[79]:
```
array([0.00840445, 0.0084857 , 0.00841868, ..., 0.02738999, 0.02715904,
       0.02676144])
```

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, $\tilde{r}_p$.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [80]:
```
#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[80]:
```
array([-0.01262911,  0.00056176, -0.0061875 , ..., -0.00414611,
        0.00514476, -0.00946444])
```

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 [81]:
```
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[81]:

	Standardized Outcome
126	-1.502669
127	0.066201
128	-0.734973
129	-0.016849
130	0.009584
...	...
1506	0.440712
1507	0.189176
1508	-0.151373
1509	0.189431
1510	-0.353660

1385 rows × 1 columns

Rolling Window 63

Loading [MathJax]/jax/output/HTML-CSS/jax.js

i) Generate a covariance matrixes for generated portfolio.

In [82]:
```
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[82]:

	Ticker #	1	2	3	4	5	6	7	8
	Ticker #								
62	1	0.000632	0.000198	0.000263	0.000097	0.000066	0.000207	0.000192	0.000119
	2	0.000198	0.000391	0.000206	-0.000027	0.000041	0.000140	0.000150	0.000113
	3	0.000263	0.000206	0.001327	0.000140	0.000238	0.000196	0.000387	0.000139
	4	0.000097	-0.000027	0.000140	0.000626	0.000007	-0.000044	0.000040	0.000023
	5	0.000066	0.000041	0.000238	0.000007	0.000237	0.000092	0.000085	0.000040
...	...	...	...	...	...	...	...	...	...
1509	96	0.000241	0.000316	0.000219	0.000169	0.000143	0.000141	0.000228	0.000099
	97	0.000248	0.000427	0.000427	0.000238	0.000290	0.000161	0.000354	0.000147
	98	0.000271	0.000581	0.001058	0.000278	0.000306	0.000132	0.000350	0.000154
	99	0.000113	0.000208	0.000064	0.000106	0.000098	0.000015	0.000145	0.000074
	100	0.000247	0.000608	0.000099	0.000176	0.000212	0.000252	0.000140	0.000189

144800 rows × 100 columns

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

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [83]:
```
#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[83]:

Ticker #	1	2	3	4	5	6	7	8	9
0	0.018568	0.043998	0.001397	0.000321	0.004069	0.010625	0.000441	0.008199	0.00021[illegible]
1	0.018809	0.044861	0.001390	0.000318	0.003980	0.010929	0.000403	0.008369	0.000217
2	0.019375	0.044465	0.001475	0.000322	0.004046	0.010875	0.000442	0.008323	0.00021[illegible]
3	0.019478	0.043805	0.001497	0.000318	0.004142	0.011039	0.000466	0.008342	0.00021[illegible]
4	0.020277	0.044135	0.001391	0.000317	0.004015	0.011364	0.000467	0.008265	0.00021[illegible]
...	...	...	...	...	...	...	...	...	...
1505	0.013604	0.020282	0.000935	0.000440	0.005093	0.236809	0.000689	0.009864	0.00012[illegible]
1506	0.013457	0.020469	0.000932	0.000440	0.005034	0.242061	0.000694	0.009756	0.00011[illegible]
1507	0.013420	0.020067	0.000895	0.000433	0.004990	0.243796	0.000701	0.009732	0.00012[illegible]
1508	0.013447	0.020065	0.000892	0.000441	0.004955	0.241941	0.000713	0.009759	0.00012[illegible]
1509	0.013719	0.020300	0.000894	0.000437	0.004960	0.243645	0.000721	0.009688	0.00012[illegible]

1510 rows × 100 columns

In [84]:
```
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[84]:
```
array([0.00885192, 0.00908816, 0.00906737, ..., 0.01689205, 0.01687095,
       0.01685115])
```

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, $\tilde{r}_p$.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [85]:
```
#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[85]:
```
array([ 0.01740121,  0.0116785 , -0.00496981, ..., -0.00414611,
        0.00514476, -0.00946444])
```

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 [86]:
```
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[86]:

	Standardized Outcome
63	1.965812
64	1.285023
65	-0.548099
66	-0.525563
67	0.197667
...	...
1506	0.706105
1507	0.302347
1508	-0.245447
1509	0.304948
1510	-0.561650

1448 rows × 1 columns

b) Compute bias statistics.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [87]:
```
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)
```

```
1.1687924023168035 1.0366178517316669 0.9687974333689725 0.946844788289
987
```

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

Question 3

We now consider the market model approach to estimating volatility. Each step below shouldbe completed using 504, 252, 126, and 63 day rolling windows.

Portfolio from part 2:

In [88]:
```
portfolio_q2_ret
```

Out[88]:

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 × 100 columns

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [89]:
```
portfolio_q2_cap
```

Out[89]:

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 × 100 columns

In [90]:
```
df_weights = df_weights_q2_504
df_weights
```

Out[90]:

Ticker	1	2	3	4	5	6	7	8	9
#									
0	0.018568	0.043998	0.001397	0.000321	0.004069	0.010625	0.000441	0.008199	0.000216
1	0.018809	0.044861	0.001390	0.000318	0.003980	0.010929	0.000403	0.008369	0.000217
2	0.019375	0.044465	0.001475	0.000322	0.004046	0.010875	0.000442	0.008323	0.000218
3	0.019478	0.043805	0.001497	0.000318	0.004142	0.011039	0.000466	0.008342	0.000219
4	0.020277	0.044135	0.001391	0.000317	0.004015	0.011364	0.000467	0.008265	0.000219
...	...	...	...	...	...	...	...	...	...
1505	0.013604	0.020282	0.000935	0.000440	0.005093	0.236809	0.000689	0.009864	0.000120
1506	0.013457	0.020469	0.000932	0.000440	0.005034	0.242061	0.000694	0.009756	0.000119
1507	0.013420	0.020067	0.000895	0.000433	0.004990	0.243796	0.000701	0.009732	0.000123
1508	0.013447	0.020065	0.000892	0.000441	0.004955	0.241941	0.000713	0.009759	0.000125
1509	0.013719	0.020300	0.000894	0.000437	0.004960	0.243645	0.000721	0.009688	0.000126

1510 rows × 100 columns

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [20]:
```
#get risk premium df
eqt_risk_prem_df = portfolio_q2_ret.sub(ffdata['Risk-free rate'], axis=0
)
eqt_risk_prem_df
```

Out[20]:

Ticker #	1	2	3	4	5	6	7	8	
0	-0.015088	-0.011882	0.030212	-0.024769	0.000206	-0.004252	-0.072456	-0.038330	0.0
1	0.026002	0.032716	0.008117	0.004714	-0.009129	0.041783	-0.072746	0.033849	0.0
2	0.031432	-0.007518	0.062258	0.014156	0.017808	-0.003648	0.097004	-0.004160	0.0
3	0.012755	-0.007574	0.022808	-0.005224	0.031377	0.022595	0.062643	0.009672	0.0
4	0.045635	0.012002	-0.067054	0.001002	-0.026486	0.034046	0.006024	-0.004849	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 × 100 columns

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [21]:
```
mkt_risk_prem_df = pd.DataFrame(ffdata['Market Returns'] - ffdata['Risk-free rate'])
mkt_risk_prem_df.rename(columns={0:"Market Risk Premium"}, inplace= True)
mkt_risk_prem_df = mkt_risk_prem_df.reindex(eqt_risk_prem_df.index)
mkt_risk_prem_df
```

Out[21]:

	Market Risk Premium
0	-0.00084
1	0.01216
2	0.00186
3	0.00266
4	0.00476
...	...
1506	0.00520
1507	0.00100
1508	-0.00090
1509	-0.00070
1510	-0.00880

1511 rows × 1 columns

$$\hat{\sigma}_p^2 = Var(\tilde{r}_p) = w^T\hat{\beta}\hat{\sigma}_M^2\hat{\beta}^T w + w^T\hat{\Delta}w$$

In [129]:
```
def var_portfolio(w, b, m, d):
    return np.dot(np.dot(np.dot(np.dot(np.transpose(w), b), m), np.transpose(b)),
                  w) + np.dot(np.dot(np.transpose(w), d), w)
```

Rolling Window 504

i) Use OLS to estimate the market betas for each stock:

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [116]:
```
betas_q3_504 = np.zeros(shape=(1007,1))
for col_index in range(eqt_risk_prem_df.shape[1]):
    ri_minus_rf = eqt_risk_prem_df.iloc[:, col_index]
    rm_minus_rf = mkt_risk_prem_df[["Market Risk Premium"]]
    col_beta = []
    for i in range(1007):
        model = OLS(ri_minus_rf[i:i+503], add_constant(rm_minus_rf[i:i+5
03]))
        res = model.fit()
        beta = res.params[1]
        col_beta.append(beta)
    betas_q3_504 = np.c_[betas_q3_504, col_beta]
betas_q3_504
```

Out[116]:
```
array([[0.        , 1.58613189, 1.40039744, ..., 2.1813825 , 1.6023306
7,
        2.27424324],
       [0.        , 1.58505739, 1.39895521, ..., 2.18106146, 1.6026596
7,
        2.27349407],
       [0.        , 1.56852779, 1.38326599, ..., 2.15678101, 1.6166927
6,
        2.2867706 ],
       ...,
       [0.        , 1.04940178, 1.71999022, ..., 1.42024318, 1.0752302
5,
        1.69069858],
       [0.        , 1.04934783, 1.71996254, ..., 1.42025387, 1.0754745
7,
        1.69115285],
       [0.        , 1.04899455, 1.72005751, ..., 1.42046858, 1.0750120
8,
        1.68990684]])
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [117]:

```
betas_df_q3_504 = pd.DataFrame(betas_q3_504).drop(0, axis = 1)
betas_df_q3_504
```

Out[117]:

	1	2	3	4	5	6	7	8	9
0	1.586132	1.400397	2.201703	1.143659	0.984847	1.567119	1.805154	0.771245	1.559725
1	1.585057	1.398955	2.201317	1.139775	0.979738	1.564692	1.801294	0.769244	1.564287
2	1.568528	1.383266	2.199448	1.143749	0.980552	1.562290	1.839123	0.762197	1.543885
3	1.565407	1.383734	2.198504	1.141111	0.979743	1.561133	1.843936	0.760247	1.543489
4	1.563436	1.384451	2.197515	1.141732	0.976663	1.560020	1.838144	0.760024	1.542143
...	...	...	...	...	...	...	...	...	...
1002	1.049304	1.718971	1.326889	1.149744	0.822715	0.948279	1.031868	0.537028	1.998436
1003	1.049207	1.719024	1.326736	1.149722	0.822565	0.948359	1.033234	0.536986	1.998578
1004	1.049402	1.719990	1.325560	1.148348	0.822522	0.949578	1.031047	0.537483	1.995442
1005	1.049348	1.719963	1.325524	1.148276	0.822531	0.949600	1.031096	0.537497	1.995745
1006	1.048995	1.720058	1.325445	1.147740	0.822563	0.949548	1.030476	0.537204	1.996368

1007 rows × 100 columns

ii) Estimate the variance of the market, $\hat{\sigma}_M^2 = Var(\tilde{r}_M)$ and the idiosyncratic variance, $\hat{\sigma}_i^2 = Var(\tilde{\epsilon}_i)$, of each security in your portfolio.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [309]:
```
#variance of market returns
var_rm_q3_504 = list(ffdata['Market Returns'].rolling(504).var().dropna
())
var_rm_q3_504 = var_rm_q3_504[:-1]
pd.DataFrame(var_rm_q3_504)
```

Out[309]:

	0
0	0.000046
1	0.000047
2	0.000046
3	0.000046
4	0.000046
...	...
1002	0.000484
1003	0.000484
1004	0.000484
1005	0.000484
1006	0.000484

1007 rows × 1 columns

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [50]:
```
var_e_q3_504 = np.zeros(shape=(1007,1))
for col_index in range(eqt_risk_prem_df.shape[1]):
    ri_minus_rf = eqt_risk_prem_df.iloc[:, col_index]
    rm_minus_rf = mkt_risk_prem_df[["Market Risk Premium"]]
    col_vars = []
    for i in range(1007):
        model = OLS(ri_minus_rf[i:i+503], add_constant(rm_minus_rf[i:i+5
03]))
        res = model.fit()
        varis = res.resid
        col_vars.append(np.var(varis))
    var_e_q3_504 = np.c_[var_e_q3_504, col_vars]
var_e_q3_504
```

Out[50]:
```
array([[0.        , 0.00032241, 0.00016527, ..., 0.00090192, 0.0007921
4,
        0.00088757],
       [0.        , 0.00032204, 0.00016512, ..., 0.00090189, 0.0007920
7,
        0.00088745],
       [0.        , 0.00032268, 0.0001648 , ..., 0.00090647, 0.0007862
8,
        0.00088911],
       ...,
       [0.        , 0.00027802, 0.00092841, ..., 0.00367989, 0.0011591
9,
        0.00199991],
       [0.        , 0.00027791, 0.00092886, ..., 0.00368292, 0.0011573
3,
        0.00199361],
       [0.        , 0.00027757, 0.00092886, ..., 0.0036828 , 0.0011573
5,
        0.00198985]])
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [52]:
```
vars_df_q3_504 = pd.DataFrame(var_e_q3_504).drop(0, axis = 1)
vars_df_q3_504
```

Out[52]:

	1	2	3	4	5	6	7	8	9
0	0.000322	0.000165	0.000687	0.000422	0.000270	0.000510	0.000968	0.000170	0.000248
1	0.000322	0.000165	0.000685	0.000421	0.000271	0.000510	0.000958	0.000167	0.000249
2	0.000323	0.000165	0.000685	0.000421	0.000271	0.000510	0.000940	0.000166	0.000251
3	0.000321	0.000165	0.000679	0.000421	0.000271	0.000510	0.000929	0.000166	0.000251
4	0.000322	0.000165	0.000679	0.000421	0.000269	0.000510	0.000922	0.000166	0.000251
...	...	...	...	...	...	...	...	...	...
1002	0.000278	0.000929	0.003222	0.000736	0.000539	0.000473	0.001003	0.000275	0.002759
1003	0.000278	0.000928	0.003221	0.000733	0.000539	0.000473	0.001009	0.000275	0.002759
1004	0.000278	0.000928	0.003220	0.000731	0.000539	0.000475	0.001006	0.000275	0.002754
1005	0.000278	0.000929	0.003222	0.000732	0.000539	0.000475	0.001006	0.000275	0.002755
1006	0.000278	0.000929	0.003222	0.000731	0.000539	0.000475	0.001006	0.000275	0.002754

1007 rows × 100 columns

iii) Using the market capitalization weights (from the last day in the rolling window)of your securities, estimate the variance and standard deviation of your portfolio.

Formula:

$$\hat{\sigma}_p^2 = Var(\tilde{r}_p) = w^T \hat{\beta} \hat{\sigma}_M^2 \hat{\beta}^T w + w^T \hat{\Delta} w$$

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [99]:
```
df_weights_504 = df_weights.tail(1007)
df_weights_504
```

Out[99]:

Ticker \#	1	2	3	4	5	6	7	8	9
503	0.018565	0.027973	0.001530	0.000238	0.005111	0.066085	0.000360	0.009388	0.00027
504	0.018351	0.027784	0.001531	0.000238	0.005027	0.067496	0.000358	0.009413	0.00027
505	0.018299	0.027789	0.001563	0.000236	0.005035	0.067324	0.000378	0.009300	0.00027
506	0.018792	0.028031	0.001533	0.000237	0.005063	0.066842	0.000377	0.009191	0.000277
507	0.018690	0.027641	0.001549	0.000233	0.005003	0.067839	0.000374	0.008993	0.00027
...	...	...	...	...	...	...	...	...	...
1505	0.013604	0.020282	0.000935	0.000440	0.005093	0.236809	0.000689	0.009864	0.00012
1506	0.013457	0.020469	0.000932	0.000440	0.005034	0.242061	0.000694	0.009756	0.00011
1507	0.013420	0.020067	0.000895	0.000433	0.004990	0.243796	0.000701	0.009732	0.00012
1508	0.013447	0.020065	0.000892	0.000441	0.004955	0.241941	0.000713	0.009759	0.00012
1509	0.013719	0.020300	0.000894	0.000437	0.004960	0.243645	0.000721	0.009688	0.00012

1007 rows × 100 columns

In [133]:
```
var_port_q3_504 = []
for i in range(len(df_weights_504)):
    diag_mat = np.diag(vars_df_q3_504.iloc[i])
    res = var_portfolio(df_weights_504.iloc[i], betas_df_q3_504.iloc[i],
var_rm_q3_504[i], diag_mat)
    var_port_q3_504.append(res)
arr_var_q3_504 = np.array(var_port_q3_504)
arr_var_q3_504
```

Out[133]:
```
array([6.63549209e-05, 6.72981929e-05, 6.70513956e-05, ...,
       4.47347672e-04, 4.46184599e-04, 4.46173349e-04])
```

In [134]:
```
arr_sd_q3_504 = np.sqrt(arr_var_q3_504)
arr_sd_q3_504
```

Out[134]:
```
array([0.00814585, 0.00820355, 0.00818849, ..., 0.0211506 , 0.02112308,
       0.02112282])
```

iv) 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, $\tilde{r}_p$.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [131]:
```
#getting one-day ahead returns array
dayahead504_port_ret_q3 = []
dayahead504_ret_q3 = portfolio_q2_ret.loc[504:1510].to_numpy()
dayahead504_w_q3 = weights_q2_504.loc[504:1510].to_numpy()
for i in range(0, len(dayahead504_w_q2)):
    dayahead504_port_ret_q3.append(np.multiply(dayahead504_ret_q3[i], da
yahead504_w_q3[i]))
dayahead504_port_ret_q3_arr = np.sum(np.array(dayahead504_port_ret_q3),
axis = 1)
dayahead504_port_ret_q3_arr
```

Out[131]:
```
array([ 0.01826034,  0.00564215, -0.00055066, ..., -0.00414611,
        0.00514476, -0.00946444])
```

v) 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 [147]:
```
standardized_outcomes_504_q3 = dayahead504_port_ret_q3_arr / arr_sd_q3_5
04
std_outcomes_504_q3 = pd.DataFrame(standardized_outcomes_504_q3)
std_outcomes_504_q3.index += 504
std_outcomes_504_q3.rename(columns={0: "Standardized Outcome"}, inplace
= True)
std_outcomes_504_q3
```

Out[147]:

	Standardized Outcome
504	2.241673
505	0.687769
506	-0.067248
507	1.343726
508	0.453598
...	...
1506	0.572534
1507	0.245258
1508	-0.196028
1509	0.243561
1510	-0.448067

1007 rows × 1 columns

Rolling Window 252

Loading [MathJax]/jax/output/HTML-CSS/jax.js

i) Use OLS to estimate the market betas for each stock:

In [137]:
```
betas_q3_252 = np.zeros(shape=(1259,1))
for col_index in range(eqt_risk_prem_df.shape[1]):
    ri_minus_rf = eqt_risk_prem_df.iloc[:, col_index]
    rm_minus_rf = mkt_risk_prem_df[["Market Risk Premium"]]
    col_beta = []
    for i in range(1259):
        model = OLS(ri_minus_rf[i:i+251], add_constant(rm_minus_rf[i:i+2
51]))
        res = model.fit()
        beta = res.params[1]
        col_beta.append(beta)
    betas_q3_252 = np.c_[betas_q3_252, col_beta]
betas_q3_252
```

Out[137]:
```
array([[0.        , 1.9319157 , 1.56218804, ..., 2.6725418 , 1.8120748
6,
         2.30861338],
       [0.        , 1.93020096, 1.56099466, ..., 2.67307771, 1.8153182
3,
         2.3086432 ],
       [0.        , 1.91678797, 1.5441574 , ..., 2.66836787, 1.8508778
7,
         2.30164876],
       ...,
       [0.        , 1.13115708, 1.93123882, ..., 1.57264432, 1.1709713
4,
         1.68377417],
       [0.        , 1.12920755, 1.93157658, ..., 1.57335547, 1.1702091
4,
         1.67916135],
       [0.        , 1.14018041, 1.91684191, ..., 1.58306463, 1.1652043
9,
         1.69638467]])
```

In [138]:
```
betas_df_q3_252 = pd.DataFrame(betas_q3_252).drop(0, axis = 1)
betas_df_q3_252
```

Out[138]:

	1	2	3	4	5	6	7	8	9
0	1.931916	1.562188	2.317098	1.147359	0.955836	1.385805	2.060276	0.899883	1.342231
1	1.930201	1.560995	2.321713	1.143028	0.955016	1.386188	2.056838	0.895685	1.346941
2	1.916788	1.544157	2.322160	1.195700	0.968974	1.372672	2.155587	0.874056	1.352562
3	1.913509	1.542065	2.354886	1.184452	0.960986	1.348913	2.160270	0.871213	1.352927
4	1.905218	1.541584	2.366488	1.183212	0.951547	1.339436	2.155961	0.868018	1.363342
...	...	...	...	...	...	...	...	...	...
1254	1.129996	1.931494	1.456411	1.198005	0.573958	0.833084	0.977473	0.467603	2.888859
1255	1.130210	1.932652	1.455659	1.196772	0.573242	0.833962	0.980778	0.467705	2.887693
1256	1.131157	1.931239	1.455531	1.197119	0.573145	0.835204	0.977920	0.467129	2.889860
1257	1.129208	1.931577	1.453296	1.198668	0.573467	0.835939	0.976104	0.466884	2.887394
1258	1.140180	1.916842	1.456712	1.199883	0.564602	0.844310	0.976781	0.461114	2.883868

1259 rows × 100 columns

ii) Estimate the variance of the market, $\hat{\sigma}_M^2 = Var(\tilde{r}_M)$ and the idiosyncratic variance, $\hat{\sigma}_i^2 = Var(\tilde{\epsilon}_i)$, of each security in your portfolio.

In [301]:
```
var_rm_q3_252 = list(ffdata['Market Returns'].rolling(252).var().dropna
())
var_rm_q3_252 = var_rm_q3_252[:-1]
np.array(var_rm_q3_252)
```

Out[301]:
```
array([5.00546569e-05, 5.04913241e-05, 5.06362050e-05, ...,
       3.19893436e-04, 3.19774427e-04, 3.17592849e-04])
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [140]:
```
var_e_q3_252 = np.zeros(shape=(1259,1))
for col_index in range(eqt_risk_prem_df.shape[1]):
    ri_minus_rf = eqt_risk_prem_df.iloc[:, col_index]
    rm_minus_rf = mkt_risk_prem_df[["Market Risk Premium"]]
    col_vars = []
    for i in range(1259):
        model = OLS(ri_minus_rf[i:i+251], add_constant(rm_minus_rf[i:i+2
51]))
        res = model.fit()
        varis = res.resid
        col_vars.append(np.var(varis))
    var_e_q3_252 = np.c_[var_e_q3_252, col_vars]
var_e_q3_252
```

Out[140]:
```
array([[0.        , 0.00039619, 0.00020445, ..., 0.00094436, 0.0005457
4,
        0.00077488],
       [0.        , 0.00039566, 0.00020412, ..., 0.00094415, 0.0005471
4,
        0.00077478],
       [0.        , 0.00039614, 0.00020326, ..., 0.0009464 , 0.0005350
6,
        0.00077391],
       ...,
       [0.        , 0.00027243, 0.00088567, ..., 0.0024766 , 0.0007879
2,
        0.00138948],
       [0.        , 0.00026971, 0.00088678, ..., 0.00248556, 0.0007875
1,
        0.00137423],
       [0.        , 0.00026425, 0.00087663, ..., 0.00248117, 0.0007877
3,
        0.001363  ]])
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [141]:
```
vars_df_q3_252 = pd.DataFrame(var_e_q3_252).drop(0, axis = 1)
vars_df_q3_252
```

Out[141]:

	1	2	3	4	5	6	7	8	9
0	0.000396	0.000204	0.000711	0.000472	0.000259	0.000552	0.001117	0.000185	0.000216
1	0.000396	0.000204	0.000707	0.000470	0.000259	0.000552	0.001101	0.000179	0.000218
2	0.000396	0.000203	0.000707	0.000481	0.000258	0.000551	0.001063	0.000177	0.000218
3	0.000393	0.000203	0.000699	0.000481	0.000257	0.000553	0.001029	0.000177	0.000218
4	0.000393	0.000203	0.000701	0.000481	0.000254	0.000552	0.001016	0.000177	0.000220
...	...	...	...	...	...	...	...	...	...
1254	0.000274	0.000891	0.002006	0.000372	0.000411	0.000232	0.000821	0.000233	0.002552
1255	0.000274	0.000888	0.002007	0.000371	0.000410	0.000230	0.000833	0.000233	0.002547
1256	0.000272	0.000886	0.002007	0.000371	0.000410	0.000233	0.000824	0.000232	0.002542
1257	0.000270	0.000887	0.002008	0.000370	0.000410	0.000233	0.000823	0.000232	0.002542
1258	0.000264	0.000877	0.002007	0.000371	0.000407	0.000231	0.000823	0.000231	0.002543

1259 rows × 100 columns

iii) Using the market capitalization weights (from the last day in the rolling window)of your securities, estimate the variance and standard deviation of your portfolio.

Formula:

$$\hat{\sigma}_p^2 = Var(\tilde{r}_p) = w^T \hat{\beta} \hat{\sigma}_M^2 \hat{\beta}^T w + w^T \hat{\Delta} w$$

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [142]:
```
df_weights_252 = df_weights.tail(1259)
df_weights_252
```

Out[142]:

Ticker #	1	2	3	4	5	6	7	8	9
251	0.014672	0.034202	0.001147	0.000248	0.004016	0.032580	0.000359	0.007714	0.00025[illegible]
252	0.014634	0.033957	0.001150	0.000233	0.004031	0.032230	0.000350	0.007694	0.00025[illegible]
253	0.014417	0.033738	0.001083	0.000234	0.004051	0.032946	0.000339	0.007724	0.000252
254	0.014450	0.033630	0.001037	0.000234	0.004070	0.033326	0.000330	0.007730	0.000244
255	0.014070	0.033624	0.001045	0.000230	0.004048	0.033202	0.000328	0.007761	0.00024[illegible]
...	...	...	...	...	...	...	...	...	...
1505	0.013604	0.020282	0.000935	0.000440	0.005093	0.236809	0.000689	0.009864	0.00012[illegible]
1506	0.013457	0.020469	0.000932	0.000440	0.005034	0.242061	0.000694	0.009756	0.00011[illegible]
1507	0.013420	0.020067	0.000895	0.000433	0.004990	0.243796	0.000701	0.009732	0.00012[illegible]
1508	0.013447	0.020065	0.000892	0.000441	0.004955	0.241941	0.000713	0.009759	0.00012[illegible]
1509	0.013719	0.020300	0.000894	0.000437	0.004960	0.243645	0.000721	0.009688	0.00012[illegible]

1259 rows × 100 columns

In [143]:
```
var_port_q3_252 = []
for i in range(len(df_weights_252)):
    diag_mat = np.diag(vars_df_q3_252.iloc[i])
    res = var_portfolio(df_weights_252.iloc[i], betas_df_q3_252.iloc[i],
var_rm_q3_252[i], diag_mat)
    var_port_q3_252.append(res)
arr_var_q3_252 = np.array(var_port_q3_252)
arr_var_q3_252
```

Out[143]:
```
array([7.10001196e-05, 7.15088950e-05, 7.13106077e-05, ...,
       2.80209391e-04, 2.79455380e-04, 2.78235687e-04])
```

In [144]:
```
arr_sd_q3_252 = np.sqrt(arr_var_q3_252)
arr_sd_q3_252
```

Out[144]:
```
array([0.00842616, 0.00845629, 0.00844456, ..., 0.01673946, 0.01671692,
       0.0166804 ])
```

iv) 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, $\tilde{r}_p$.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [145]:
```
#getting one-day ahead returns array
dayahead252_port_ret_q3 = []
dayahead252_ret_q3 = portfolio_q2_ret.loc[252:1510].to_numpy()
dayahead252_w_q3 = weights_q2_252.loc[252:1510].to_numpy()
for i in range(0, len(dayahead252_w_q2)):
    dayahead252_port_ret_q3.append(np.multiply(dayahead252_ret_q3[i], da
yahead252_w_q3[i]))
dayahead252_port_ret_q3_arr = np.sum(np.array(dayahead252_port_ret_q3),
axis = 1)
dayahead252_port_ret_q3_arr
```

Out[145]:
```
array([-0.00643605, -0.01146078, -0.00250408, ..., -0.00414611,
        0.00514476, -0.00946444])
```

v) 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 [146]:
```
standardized_outcomes_252_q3 = dayahead252_port_ret_q3_arr / arr_sd_q3_2
52
std_outcomes_252_q3 = pd.DataFrame(standardized_outcomes_252_q3)
std_outcomes_252_q3.index += 252
std_outcomes_252_q3.rename(columns={0: "Standardized Outcome"}, inplace
= True)
std_outcomes_252_q3
```

Out[146]:

	Standardized Outcome
252	-0.763818
253	-1.355296
254	-0.296531
255	0.545255
256	0.646344
...	...
1506	0.722503
1507	0.309746
1508	-0.247685
1509	0.307758
1510	-0.567399

1259 rows × 1 columns

Rolling Window 126

Loading [MathJax]/jax/output/HTML-CSS/jax.js

i) Use OLS to estimate the market betas for each stock:

In [148]:
```
betas_q3_126 = np.zeros(shape=(1385,1))
for col_index in range(eqt_risk_prem_df.shape[1]):
    ri_minus_rf = eqt_risk_prem_df.iloc[:, col_index]
    rm_minus_rf = mkt_risk_prem_df[["Market Risk Premium"]]
    col_beta = []
    for i in range(1385):
        model = OLS(ri_minus_rf[i:i+125], add_constant(rm_minus_rf[i:i+1
25]))
        res = model.fit()
        beta = res.params[1]
        col_beta.append(beta)
    betas_q3_126 = np.c_[betas_q3_126, col_beta]
betas_q3_126
```

Out[148]:
```
array([[0.        , 1.8587196 , 1.74873984, ..., 2.68095759, 1.5151042
9,
        2.36746127],
       [0.        , 1.85954524, 1.74715317, ..., 2.67584122, 1.5196995
8,
        2.37150663],
       [0.        , 1.87006592, 1.71116045, ..., 2.74035345, 1.6172243
2,
        2.33411603],
       ...,
       [0.        , 1.18005442, 1.8923816 , ..., 1.65167118, 0.9846047
7,
        2.01873032],
       [0.        , 1.181316  , 1.8953813 , ..., 1.62771547, 0.9696146
3,
        2.03000054],
       [0.        , 1.17989873, 1.89854685, ..., 1.6231458 , 0.9557467
7,
        2.03780855]])
```

In [149]:
```
betas_df_q3_126 = pd.DataFrame(betas_q3_126).drop(0, axis = 1)
betas_df_q3_126
```

Out[149]:

	1	2	3	4	5	6	7	8	9
0	1.858720	1.748740	2.090839	0.863111	0.797002	1.536315	2.082316	0.950925	1.132253
1	1.859545	1.747153	2.090576	0.861535	0.799008	1.546799	2.076093	0.945995	1.132293
2	1.870066	1.711160	2.140906	0.853506	0.832043	1.503603	2.209823	0.932673	1.136640
3	1.860781	1.719649	2.121365	0.852611	0.824514	1.499804	2.184585	0.932889	1.142616
4	1.855515	1.710480	2.139003	0.877110	0.857206	1.489830	2.152956	0.922099	1.178655
...	...	...	...	...	...	...	...	...	...
1380	1.177943	1.872456	1.285212	0.903495	0.548774	0.877056	1.121534	0.447239	1.635484
1381	1.178430	1.873308	1.284081	0.902287	0.547696	0.878505	1.132335	0.446102	1.637488
1382	1.180054	1.892382	1.269821	0.900144	0.550474	0.888848	1.134361	0.449499	1.629157
1383	1.181316	1.895381	1.273381	0.885996	0.552735	0.893454	1.139273	0.439229	1.626673
1384	1.179899	1.898547	1.262461	0.873377	0.551200	0.896687	1.144386	0.433780	1.616631

1385 rows × 100 columns

ii) Estimate the variance of the market, $\hat{\sigma}_M^2 = Var(\tilde{r}_M)$ and the idiosyncratic variance, $\hat{\sigma}_i^2 = Var(\tilde{\epsilon}_i)$, of each security in your portfolio.

In [302]:
```
var_rm_q3_126 = list(ffdata['Market Returns'].rolling(126).var().dropna
())
var_rm_q3_126 = var_rm_q3_126[:-1]
np.array(var_rm_q3_126)
```

Out[302]:
```
array([5.67710298e-05, 5.74354641e-05, 5.62886000e-05, ...,
       1.29130537e-04, 1.28452760e-04, 1.28360577e-04])
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [151]:
```
var_e_q3_126 = np.zeros(shape=(1385,1))
for col_index in range(eqt_risk_prem_df.shape[1]):
    ri_minus_rf = eqt_risk_prem_df.iloc[:, col_index]
    rm_minus_rf = mkt_risk_prem_df[["Market Risk Premium"]]
    col_vars = []
    for i in range(1385):
        model = OLS(ri_minus_rf[i:i+125], add_constant(rm_minus_rf[i:i+1
25]))
        res = model.fit()
        varis = res.resid
        col_vars.append(np.var(varis))
    var_e_q3_126 = np.c_[var_e_q3_126, col_vars]
var_e_q3_126
```

Out[151]:
```
array([[0.        , 0.00033091, 0.00023457, ..., 0.00089622, 0.0006101
7,
         0.00083786],
       [0.        , 0.00033016, 0.00023392, ..., 0.00089916, 0.0006110
3,
         0.00083976],
       [0.        , 0.0003314 , 0.00023348, ..., 0.00089376, 0.0005843
4,
         0.00083603],
       ...,
       [0.        , 0.00014199, 0.00045503, ..., 0.00159275, 0.0005720
9,
         0.00125177],
       [0.        , 0.00014195, 0.0004573 , ..., 0.00159915, 0.0005670
3,
         0.00124892],
       [0.        , 0.00014161, 0.00045658, ..., 0.00159687, 0.0005547
2,
         0.0012489 ]])
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [152]:
```
vars_df_q3_126 = pd.DataFrame(var_e_q3_126).drop(0, axis = 1)
vars_df_q3_126
```

Out[152]:

	1	2	3	4	5	6	7	8	9
0	0.000331	0.000235	0.000796	0.000553	0.000191	0.000413	0.001187	0.000131	0.000167
1	0.000330	0.000234	0.000791	0.000548	0.000191	0.000424	0.001148	0.000119	0.000166
2	0.000331	0.000233	0.000789	0.000550	0.000188	0.000421	0.001074	0.000118	0.000166
3	0.000325	0.000237	0.000765	0.000549	0.000188	0.000425	0.001007	0.000118	0.000171
4	0.000325	0.000237	0.000766	0.000551	0.000190	0.000423	0.000982	0.000118	0.000177
...	...	...	...	...	...	...	...	...	...
1380	0.000145	0.000470	0.001286	0.000267	0.000243	0.000147	0.000671	0.000142	0.001309
1381	0.000142	0.000470	0.001285	0.000267	0.000243	0.000145	0.000701	0.000141	0.001309
1382	0.000142	0.000455	0.001271	0.000266	0.000242	0.000150	0.000702	0.000140	0.001305
1383	0.000142	0.000457	0.001280	0.000262	0.000242	0.000150	0.000703	0.000138	0.001314
1384	0.000142	0.000457	0.001265	0.000253	0.000242	0.000151	0.000698	0.000134	0.001310

1385 rows × 100 columns

iii) Using the market capitalization weights (from the last day in the rolling window)of your securities, estimate the variance and standard deviation of your portfolio.

Formula:

$$\hat{\sigma}_p^2 = Var(\tilde{r}_p) = w^T\hat{\beta}\hat{\sigma}_M^2\hat{\beta}^T w + w^T\hat{\Delta}w$$

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [153]:
```
df_weights_126 = df_weights.tail(1385)
df_weights_126
```

Out[153]:

Ticker #	1	2	3	4	5	6	7	8	9
125	0.018116	0.037240	0.001599	0.000318	0.004287	0.015935	0.000463	0.008304	0.000257
126	0.017800	0.037490	0.001569	0.000324	0.004315	0.016075	0.000474	0.008177	0.000257
127	0.017727	0.038398	0.001550	0.000328	0.004267	0.015777	0.000474	0.008146	0.000264
128	0.017547	0.038386	0.001501	0.000322	0.004119	0.015748	0.000472	0.008176	0.000255
129	0.017530	0.038236	0.001515	0.000323	0.004116	0.015692	0.000479	0.008088	0.000256
...	...	...	...	...	...	...	...	...	...
1505	0.013604	0.020282	0.000935	0.000440	0.005093	0.236809	0.000689	0.009864	0.000120
1506	0.013457	0.020469	0.000932	0.000440	0.005034	0.242061	0.000694	0.009756	0.000119
1507	0.013420	0.020067	0.000895	0.000433	0.004990	0.243796	0.000701	0.009732	0.000123
1508	0.013447	0.020065	0.000892	0.000441	0.004955	0.241941	0.000713	0.009759	0.000125
1509	0.013719	0.020300	0.000894	0.000437	0.004960	0.243645	0.000721	0.009688	0.000126

1385 rows × 100 columns

In [154]:
```
var_port_q3_126 = []
for i in range(len(df_weights_126)):
    diag_mat = np.diag(vars_df_q3_126.iloc[i])
    res = var_portfolio(df_weights_126.iloc[i], betas_df_q3_126.iloc[i],
var_rm_q3_126[i], diag_mat)
    var_port_q3_126.append(res)
arr_var_q3_126 = np.array(var_port_q3_126)
arr_var_q3_126
```

Out[154]:
```
array([6.97628889e-05, 7.02152997e-05, 6.97454828e-05, ...,
       1.08765762e-04, 1.07983400e-04, 1.08278851e-04])
```

In [155]:
```
arr_sd_q3_126 = np.sqrt(arr_var_q3_126)
arr_sd_q3_126
```

Out[155]:
```
array([0.00835242, 0.00837946, 0.00835138, ..., 0.01042908, 0.01039151,
       0.01040571])
```

iv) 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, $\tilde{r}_p$.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [156]:
```
#getting one-day ahead returns array
dayahead126_port_ret_q3 = []
dayahead126_ret_q3 = portfolio_q2_ret.loc[126:1510].to_numpy()
dayahead126_w_q3 = weights_q2_126.loc[126:1510].to_numpy()
for i in range(0, len(dayahead126_w_q2)):
    dayahead126_port_ret_q3.append(np.multiply(dayahead126_ret_q3[i], da
yahead126_w_q3[i]))
dayahead126_port_ret_q3_arr = np.sum(np.array(dayahead126_port_ret_q3),
axis = 1)
dayahead126_port_ret_q3_arr
```

Out[156]:
```
array([-0.01262911,  0.00056176, -0.0061875 , ..., -0.00414611,
        0.00514476, -0.00946444])
```

v) 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 [158]:
```
standardized_outcomes_126_q3 = dayahead126_port_ret_q3_arr / arr_sd_q3_1
26
std_outcomes_126_q3 = pd.DataFrame(standardized_outcomes_126_q3)
std_outcomes_126_q3.index += 126
std_outcomes_126_q3.rename(columns={0: "Standardized Outcome"}, inplace
= True)
std_outcomes_126_q3
```

Out[158]:

	Standardized Outcome
126	-1.512030
127	0.067040
128	-0.740896
129	-0.016961
130	0.009658
...	...
1506	1.164179
1507	0.498764
1508	-0.397552
1509	0.495093
1510	-0.909543

1385 rows × 1 columns

Rolling Window 63

Loading [MathJax]/jax/output/HTML-CSS/jax.js

i) Use OLS to estimate the market betas for each stock:

In [159]:
```
betas_q3_63 = np.zeros(shape=(1448,1))
for col_index in range(eqt_risk_prem_df.shape[1]):
    ri_minus_rf = eqt_risk_prem_df.iloc[:, col_index]
    rm_minus_rf = mkt_risk_prem_df[["Market Risk Premium"]]
    col_beta = []
    for i in range(1448):
        model = OLS(ri_minus_rf[i:i+62], add_constant(rm_minus_rf[i:i+62
]))
        res = model.fit()
        beta = res.params[1]
        col_beta.append(beta)
    betas_q3_63 = np.c_[betas_q3_63, col_beta]
betas_q3_63
```

Out[159]:
```
array([[0.        , 2.05281222, 1.6314848 , ..., 2.75372606, 1.2764953
8,
        2.98277006],
       [0.        , 2.02427953, 1.61056927, ..., 2.7743287 , 1.3806743
3,
        3.00345426],
       [0.        , 2.02989276, 1.61751051, ..., 2.95940282, 1.5551235
9,
        2.94348151],
       ...,
       [0.        , 1.17241229, 1.90870263, ..., 2.19046177, 0.8616258
8,
        2.04912328],
       [0.        , 1.17280462, 1.90645963, ..., 2.19202489, 0.8599611
6,
        2.04474499],
       [0.        , 1.17361875, 1.90209361, ..., 2.19155911, 0.8424587
6,
        2.10855709]])
```

In [160]:
```
betas_df_q3_63 = pd.DataFrame(betas_q3_63).drop(0, axis = 1)
betas_df_q3_63
```

Out[160]:

	1	2	3	4	5	6	7	8	9
0	2.052812	1.631485	2.467977	0.149854	0.834601	1.489859	2.200903	0.891679	1.255516
1	2.024280	1.610569	2.470555	0.134584	0.813522	1.469599	2.103833	0.854639	1.232819
2	2.029893	1.617511	2.626662	0.150088	0.836575	1.398232	2.441186	0.785010	1.221262
3	2.002958	1.605841	2.581234	0.202040	0.870995	1.430943	2.490066	0.782768	1.185146
4	1.986856	1.585203	2.553937	0.221884	0.851413	1.438855	2.489432	0.773026	1.202064
...	...	...	...	...	...	...	...	...	...
1443	1.155123	1.901754	1.162571	0.910696	0.745226	0.835353	1.144480	0.652799	1.932211
1444	1.152748	1.883088	1.184261	0.903087	0.751742	0.834111	1.189235	0.650191	1.972411
1445	1.172412	1.908703	1.270773	0.916972	0.768153	0.839263	1.201479	0.652601	2.109775
1446	1.172805	1.906460	1.270128	0.914495	0.766944	0.837897	1.198084	0.654865	2.110638
1447	1.173619	1.902094	1.272323	0.905755	0.772203	0.841758	1.190653	0.665431	2.118114

1448 rows × 100 columns

ii) Estimate the variance of the market, $\hat{\sigma}_M^2 = Var(\tilde{r}_M)$ and the idiosyncratic variance, $\hat{\sigma}_i^2 = Var(\tilde{\epsilon}_i)$, of each security in your portfolio.

In [303]:
```
var_rm_q3_63 = list(ffdata['Market Returns'].rolling(63).var().dropna())
var_rm_q3_63 = var_rm_q3_63[:-1]
np.array(var_rm_q3_63)
```

Out[303]:
```
array([5.84349053e-05, 5.94350691e-05, 5.78318894e-05, ...,
       1.22108331e-04, 1.22095120e-04, 1.21878664e-04])
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [162]:
```python
var_e_q3_63 = np.zeros(shape=(1448,1))
for col_index in range(eqt_risk_prem_df.shape[1]):
    ri_minus_rf = eqt_risk_prem_df.iloc[:, col_index]
    rm_minus_rf = mkt_risk_prem_df[["Market Risk Premium"]]
    col_vars = []
    for i in range(1448):
        model = OLS(ri_minus_rf[i:i+62], add_constant(rm_minus_rf[i:i+62
]))
        res = model.fit()
        varis = res.resid
        col_vars.append(np.var(varis))
    var_e_q3_63 = np.c_[var_e_q3_63, col_vars]
var_e_q3_63
```

Out[162]:
```
array([[0.00000000e+00, 3.88278167e-04, 2.37273297e-04, ...,
        1.09215757e-03, 3.43703982e-04, 1.21146211e-03],
       [0.00000000e+00, 3.88152295e-04, 2.37854832e-04, ...,
        1.09327570e-03, 4.02094536e-04, 1.21369986e-03],
       [0.00000000e+00, 3.88324918e-04, 2.44060545e-04, ...,
        1.10035869e-03, 3.47754641e-04, 1.21060880e-03],
       ...,
       [0.00000000e+00, 8.87651773e-05, 3.92475094e-04, ...,
        1.01033499e-03, 2.89565134e-04, 1.13675361e-03],
       [0.00000000e+00, 8.87462678e-05, 3.96649166e-04, ...,
        1.04796946e-03, 2.89027862e-04, 1.13307292e-03],
       [0.00000000e+00, 8.87714491e-05, 3.95288783e-04, ...,
        1.04857770e-03, 2.82756791e-04, 9.51713160e-04]])
```

In [163]:
```python
vars_df_q3_63 = pd.DataFrame(var_e_q3_63).drop(0, axis = 1)
vars_df_q3_63
```

Out[163]:

	1	2	3	4	5	6	7	8	9
0	0.000388	0.000237	0.000973	0.000625	0.000196	0.000434	0.001506	0.000115	0.000176
1	0.000388	0.000238	0.000956	0.000614	0.000199	0.000435	0.001449	0.000096	0.000178
2	0.000388	0.000244	0.000970	0.000616	0.000198	0.000428	0.001276	0.000087	0.000179
3	0.000378	0.000244	0.000919	0.000632	0.000203	0.000431	0.001189	0.000087	0.000186
4	0.000379	0.000252	0.000918	0.000634	0.000189	0.000431	0.001156	0.000086	0.000191
...	...	...	...	...	...	...	...	...	...
1443	0.000090	0.000399	0.001592	0.000142	0.000184	0.000189	0.000300	0.000104	0.001438
1444	0.000090	0.000392	0.001582	0.000144	0.000183	0.000189	0.000361	0.000104	0.001416
1445	0.000089	0.000392	0.001560	0.000144	0.000182	0.000202	0.000365	0.000104	0.001352
1446	0.000089	0.000397	0.001578	0.000145	0.000182	0.000204	0.000368	0.000103	0.001369
1447	0.000089	0.000395	0.001578	0.000147	0.000184	0.000206	0.000371	0.000098	0.001365

1448 rows × 100 columns

Loading [MathJax]/jax/output/HTML-CSS/jax.js

iii) Using the market capitalization weights (from the last day in the rolling window)of your securities, estimate the variance and standard deviation of your portfolio.

Formula:

$$\hat{\sigma}_p^2 = Var(\tilde{r}_p) = w^T \hat{\beta} \hat{\sigma}_M^2 \hat{\beta}^T w + w^T \hat{\Delta} w$$

In [164]:
```
df_weights_63 = df_weights.tail(1448)
df_weights_63
```

Out[164]:

Ticker #	1	2	3	4	5	6	7	8	9
62	0.019876	0.039362	0.001319	0.000330	0.003897	0.013208	0.000505	0.007949	0.000241
63	0.019960	0.040117	0.001373	0.000328	0.003792	0.013172	0.000508	0.007900	0.000238
64	0.019807	0.039649	0.001360	0.000336	0.003860	0.013410	0.000537	0.007826	0.000233
65	0.019982	0.040530	0.001374	0.000334	0.003871	0.013246	0.000521	0.007873	0.000229
66	0.019650	0.038668	0.001337	0.000330	0.003896	0.013063	0.000520	0.007886	0.000228
...	...	...	...	...	...	...	...	...	...
1505	0.013604	0.020282	0.000935	0.000440	0.005093	0.236809	0.000689	0.009864	0.000120
1506	0.013457	0.020469	0.000932	0.000440	0.005034	0.242061	0.000694	0.009756	0.000119
1507	0.013420	0.020067	0.000895	0.000433	0.004990	0.243796	0.000701	0.009732	0.000123
1508	0.013447	0.020065	0.000892	0.000441	0.004955	0.241941	0.000713	0.009759	0.000125
1509	0.013719	0.020300	0.000894	0.000437	0.004960	0.243645	0.000721	0.009688	0.000126

1448 rows × 100 columns

In [165]:
```
var_port_q3_63 = []
for i in range(len(df_weights_63)):
    diag_mat = np.diag(vars_df_q3_63.iloc[i])
    res = var_portfolio(df_weights_63.iloc[i], betas_df_q3_63.iloc[i], v
ar_rm_q3_63[i], diag_mat)
    var_port_q3_63.append(res)
arr_var_q3_63 = np.array(var_port_q3_63)
arr_var_q3_63
```

Out[165]:
```
array([7.63172963e-05, 8.01567192e-05, 8.07554042e-05, ...,
       1.09152319e-04, 1.08705594e-04, 1.09124494e-04])
```

In [166]:
```
arr_sd_q3_63 = np.sqrt(arr_var_q3_63)
arr_sd_q3_63
```

Out[166]:
```
array([0.00873598, 0.00895303, 0.0089864 , ..., 0.0104476 , 0.0104262 ,
       0.01044627])
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

iv) 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 [260]:
```
#getting one-day ahead returns array
dayahead63_port_ret_q3 = []
dayahead63_ret_q3 = portfolio_q2_ret.loc[63:1510].to_numpy()
dayahead63_w_q3 = weights_q2_63.loc[63:1510].to_numpy()
for i in range(0, len(dayahead63_w_q2)):
    dayahead63_port_ret_q3.append(np.multiply(dayahead63_ret_q3[i], daya
head63_w_q3[i]))
dayahead63_port_ret_q3_arr = np.sum(np.array(dayahead63_port_ret_q3), ax
is = 1)
dayahead63_port_ret_q3_arr
```

Out[260]:
```
array([ 0.01740121,  0.0116785 , -0.00496981, ..., -0.00414611,
        0.00514476, -0.00946444])
```

v) 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 [168]:
```
standardized_outcomes_63_q3 = dayahead63_port_ret_q3_arr / arr_sd_q3_63
std_outcomes_63_q3 = pd.DataFrame(standardized_outcomes_63_q3)
std_outcomes_63_q3.index += 63
std_outcomes_63_q3.rename(columns={0: "Standardized Outcome"}, inplace =
True)
std_outcomes_63_q3
```

Out[168]:

	Standardized Outcome
63	1.991902
64	1.304419
65	-0.553037
66	-0.527825
67	0.198191
...	...
1506	1.133099
1507	0.485674
1508	-0.396848
1509	0.493445
1510	-0.906012

1448 rows × 1 columns

Loading [MathJax]/jax/output/HTML-CSS/jax.js

b) Compute bias statistics.

In [169]:
```
bias_stat_q3_504 = np.std(standardized_outcomes_504_q3)
bias_stat_q3_252 = np.std(standardized_outcomes_252_q3)
bias_stat_q3_126 = np.std(standardized_outcomes_126_q3)
bias_stat_q3_63 = np.std(standardized_outcomes_63_q3)
print(bias_stat_q3_504, bias_stat_q3_252, bias_stat_q3_126, bias_stat_q3
_63)
```

```
1.3118478769931892 1.1320711225951727 1.0749500838326524 1.057843577865
541
```

Question 5

In [239]:
```
#randomized portfolio indices
stocks = pd.read_csv("fifty_portfolios.csv")
stocks = stocks[0:50]
stocks.head()
```

Out[239]:

	984	1236	505	1235	1552	1732	918	863	169	1157	...	349	1588	1731	616	5
0	1757	1199	1673	1702	1511	86	1428	307	170	159	...	1044	238	1060	1646	13
1	1212	1755	1549	1121	193	1270	679	34	1816	1626	...	1247	484	141	32	5
2	425	1082	1616	784	1197	1067	215	410	1752	1081	...	1785	1797	341	997	2
3	1006	110	980	273	682	1473	887	289	989	1341	...	508	792	2	1356	13
4	26	1386	766	1435	466	1792	1865	1491	17	79	...	1577	1685	1088	247	

5 rows × 100 columns

In [240]:
```
best_rolling_window_q3 = 63
print("Best Rolling Window Market Model:", 63)
```

```
Best Rolling Window Market Model: 63
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [241]:
```
betas_df_q3_63
```

Out[241]:

	1	2	3	4	5	6	7	8	9
0	2.052812	1.631485	2.467977	0.149854	0.834601	1.489859	2.200903	0.891679	1.255516
1	2.024280	1.610569	2.470555	0.134584	0.813522	1.469599	2.103833	0.854639	1.232819
2	2.029893	1.617511	2.626662	0.150088	0.836575	1.398232	2.441186	0.785010	1.221262
3	2.002958	1.605841	2.581234	0.202040	0.870995	1.430943	2.490066	0.782768	1.185146
4	1.986856	1.585203	2.553937	0.221884	0.851413	1.438855	2.489432	0.773026	1.202064
...	...	...	...	...	...	...	...	...	...
1443	1.155123	1.901754	1.162571	0.910696	0.745226	0.835353	1.144480	0.652799	1.932211
1444	1.152748	1.883088	1.184261	0.903087	0.751742	0.834111	1.189235	0.650191	1.972411
1445	1.172412	1.908703	1.270773	0.916972	0.768153	0.839263	1.201479	0.652601	2.109775
1446	1.172805	1.906460	1.270128	0.914495	0.766944	0.837897	1.198084	0.654865	2.110638
1447	1.173619	1.902094	1.272323	0.905755	0.772203	0.841758	1.190653	0.665431	2.118114

1448 rows × 100 columns

In [242]:
```
df_weightz = secdata_cap_group.iloc[:, :].apply(lambda x: x.div(x.sum
()), axis=1)
df_weightz
```

Out[242]:

Ticker #	1	2	3	4	5	6	7	8	9
0	0.001326	0.003143	0.000100	0.000023	0.000291	0.000759	0.000031	0.000586	0.000015
1	0.001345	0.003207	0.000099	0.000023	0.000285	0.000781	0.000029	0.000598	0.000015
2	0.001385	0.003178	0.000105	0.000023	0.000289	0.000777	0.000032	0.000595	0.000016
3	0.001399	0.003145	0.000108	0.000023	0.000297	0.000793	0.000033	0.000599	0.000016
4	0.001456	0.003169	0.000100	0.000023	0.000288	0.000816	0.000034	0.000594	0.000016
...	...	...	...	...	...	...	...	...	...
1506	0.000934	0.001421	0.000065	0.000031	0.000350	0.016806	0.000048	0.000677	0.000008
1507	0.000936	0.001399	0.000062	0.000030	0.000348	0.016996	0.000049	0.000678	0.000009
1508	0.000935	0.001395	0.000062	0.000031	0.000345	0.016821	0.000050	0.000679	0.000009
1509	0.000959	0.001419	0.000062	0.000031	0.000347	0.017025	0.000050	0.000677	0.000009
1510	0.000977	0.001415	0.000063	0.000030	0.000345	0.017102	0.000051	0.000669	0.000009

1511 rows × 1877 columns

In [243]:
```
def market_beta(w, b):
    return np.dot(np.transpose(w), b)
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

a)

Date: 12/30/2005 (idx loc = 503)

In [304]:
```
betas63_q3_2005 = []
for i in range(50):
    beta_m = betas_df_q3_63.iloc[63]
    weights = df_weightz[list(stocks.iloc[i])].iloc[63]
    betas_market = market_beta(weights, beta_m)
    betas63_q3_2005.append(betas_market)
betas63_q3_2005[0:5]
```

Out[304]:
```
[0.05247658547629155 4,
 0.060517796382025806,
 0.05266254554281719,
 0.04484296288982689,
 0.05008550819633388]
```

Date: 12/31/2007 (idx loc = 1005)

In [305]:
```
betas63_q3_2007 = []
for i in range(50):
    beta_m = betas_df_q3_63.iloc[942]
    weights = df_weightz[list(stocks.iloc[i])].iloc[942]
    betas_market = market_beta(weights, beta_m)
    betas63_q3_2007.append(betas_market)
betas63_q3_2007[0:5]
```

Out[305]:
```
[0.04844186126740678,
 0.05022385833208992,
 0.04548861529658407,
 0.05461929366595343,
 0.06154826101847503 4]
```

Date: 12/31/2009 (idx loc = 1510)

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [306]:
```
betas63_q3_2009 = []
for i in range(50):
    beta_m = betas_df_q3_63.iloc[1447]
    weights = df_weightz[list(stocks.iloc[i])].iloc[1447]
    betas_market = market_beta(weights, beta_m)
    betas63_q3_2009.append(betas_market)
betas63_q3_2009[0:5]
```

Out[306]:
```
[0.041969788001929145,
 0.03609766003979558,
 0.0571643240845565,
 0.05099314794069944,
 0.06269323967843796]
```

b)

Date: 12/30/2005 (idx loc = 503)

In [293]:
```
#excludes the first 63 days
rp_2005 = 1/63*np.sum((dayahead63_port_ret_q3_arr[378:441] - np.array(ff
data['Risk-free rate'])[441:504]))
rp_2005
```

Out[293]:
```
0.0010859704382670666
```

In [290]:
```
rm_2005 = 1/63*np.sum((np.array(ffdata['Market Returns'])[441:504] - np.
array(ffdata['Risk-free rate'])[441:504]))
rm_2005
```

Out[290]:
```
8.365079365079375e-05
```

Date: 12/31/2007 (idx loc = 1005)

In [286]:
```
rp_2007 = 1/63*np.sum((dayahead63_port_ret_q3_arr[880:943] - np.array(ff
data['Risk-free rate'])[943:1006]))
rp_2007
```

Out[286]:
```
-0.00011254414214054921
```

In [291]:
```
rm_2007 = 1/63*np.sum((np.array(ffdata['Market Returns'])[943:1006] - np
.array(ffdata['Risk-free rate'])[943:1006]))
rm_2007
```

Out[291]:
```
-0.0008863492063492066
```

Date: 12/31/2009 (idx loc = 1510)

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [289]:
```
rp_2009 = 1/63*np.sum((dayahead63_port_ret_q3_arr[1385:1448] - np.array(
ffdata['Risk-free rate'])[1448:1511]))
rp_2009
```

Out[289]: 0.001998517909506608

In [292]:
```
rm_2009 = 1/63*np.sum((np.array(ffdata['Market Returns'])[1448:1511] - n
p.array(ffdata['Risk-free rate'])[1448:1511]))
rm_2009
```

Out[292]: 0.0013793650793650792

In [ ]:

Loading [MathJax]/jax/output/HTML-CSS/jax.js