econ139_final

December 18, 2020

[1]:
```
import pandas as pd
import numpy as np
import heapq

# statsmodels libraries used for computing rolling statistics.
import statsmodels.api as sm
from statsmodels.regression.rolling import RollingOLS
```

[2]:
```
# Reading in provided data into pandas DataFrames.

dates = pd.read_csv("retdate.txt", delim_whitespace=True)
secdata = pd.read_csv("secdata.txt", delim_whitespace=True)
secdata = secdata.drop(["mkt_cap"], axis=1)

mkt_cap = pd.read_csv("secdata.txt", delim_whitespace=True)
mkt_cap = mkt_cap.drop(["ret"], axis=1)

ticker_txt = open("ticker.txt", "r")
tickers = pd.Series([each.strip() for each in ticker_txt])
ff_ret = pd.read_csv("ffdata.txt", delim_whitespace=True)
ff_ret.index = pd.to_datetime(dates['date'].astype(str), format='%Y%m%d')
```

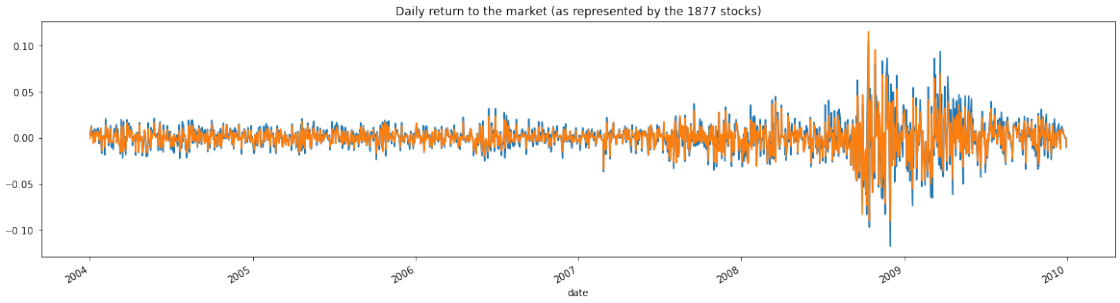
[3]:
```
return_data = pd.DataFrame()
mktcap_data = pd.DataFrame()

# Populate return and market cap data frames.
for idx, each in enumerate(tickers, start=1):
    return_data[each] = secdata.loc[secdata['idx'] == idx]["ret"].values
    mktcap_data[each] = mkt_cap.loc[mkt_cap['idx'] == idx]["mkt_cap"].values

return_data['avg'] = return_data.mean(axis=1)

# Replace indices with timestamps.
return_data.index = pd.to_datetime(dates['date'].astype(str), format='%Y%m%d')
ff_ret.index = pd.to_datetime(dates['date'].astype(str), format='%Y%m%d')
mktcap_data.index = pd.to_datetime(dates['date'].astype(str), format='%Y%m%d')
```

[4]:
```
# Helper function for plotting cumulative returns.
def cumulative_returns(ser):
    return ((ser + 1).cumprod())
```

0.0.1 1A: Plot cumulative returns to the market.

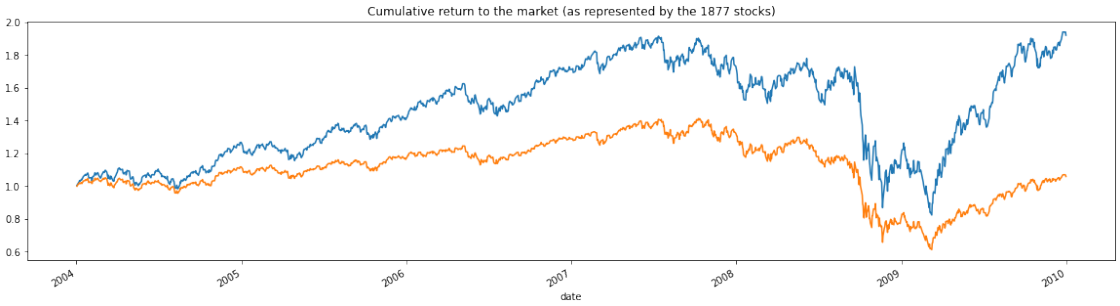
[5]:
```
# Plot of daily returns
return_data["avg"].plot(title='Daily return to the market (as represented by␣
 ↪the 1877 stocks)', figsize=(20,5))
ff_ret["mkt_ret"].plot()
```

[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2d78b69110>



[6]:
```
# Plot of cumulative returns
cumulative_returns(return_data['avg']).plot(figsize=(20,5), title='Cumulative␣
 ↪return to the market (as represented by the 1877 stocks)')
cumulative_returns(ff_ret['mkt_ret']).plot()
```

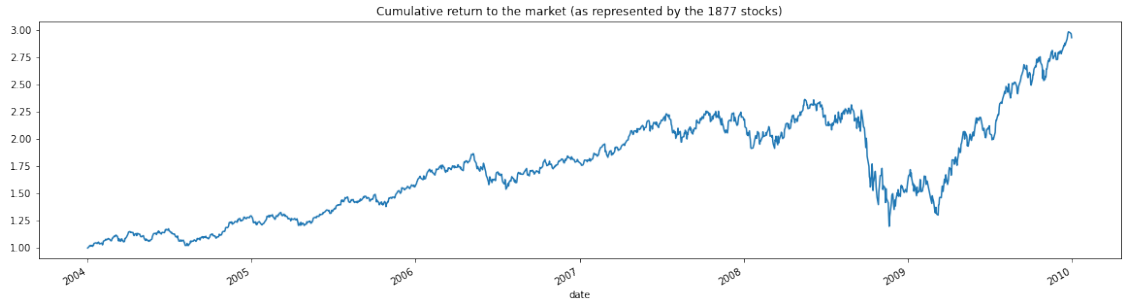
[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2d78a75090>



0.0.2 1B: Cumulative return of 50 random securities weighted by capitalization

[7]:
```
selected = tickers.sample(n=50)
cloned = mktcap_data[selected].copy()
cloned['sum'] = cloned[selected].sum(axis=1)
cloned[selected] = cloned[selected].div(cloned['sum'].values,axis=0)

weighted_returns = pd.DataFrame()
for each in selected:
    weighted_returns[each] = cloned[each] * return_data[each]

weighted_returns['summed'] = weighted_returns.sum(axis=1)
cumulative_returns(weighted_returns['summed']).plot(figsize=(20,5),␣
↪title='Cumulative return to the market (as represented by the 1877 stocks)')
```

[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2d781c8790>
```



[8]:
```
# Calculate the RMSE (root-mean-squared error) over entire sample period.
((weighted_returns['summed'] - return_data['avg']) ** 2).mean() ** .5
```

[8]:
```
0.00584743597997839
```

Can you reduce RSME by choosing a different portfolio of 50 stock?

[9]:
```
selected = tickers.sample(n=50)
cloned = mktcap_data[selected].copy()
cloned['sum'] = cloned[selected].sum(axis=1)
cloned[selected] = cloned[selected].div(cloned['sum'].values,axis=0)

weighted_returns = pd.DataFrame()
for each in selected:
    weighted_returns[each] = cloned[each] * return_data[each]

weighted_returns['summed'] = weighted_returns.sum(axis=1)
((weighted_returns['summed'] - return_data['avg']) ** 2).mean() ** .5
```

[9]: 0.005803400893666997

Yes, it is possible to reduce RSME by choosing a different random portfolio of 50 stocks.

0.0.3 1C: Pick 25 n-stock portfolios of randomly chosen stocks.

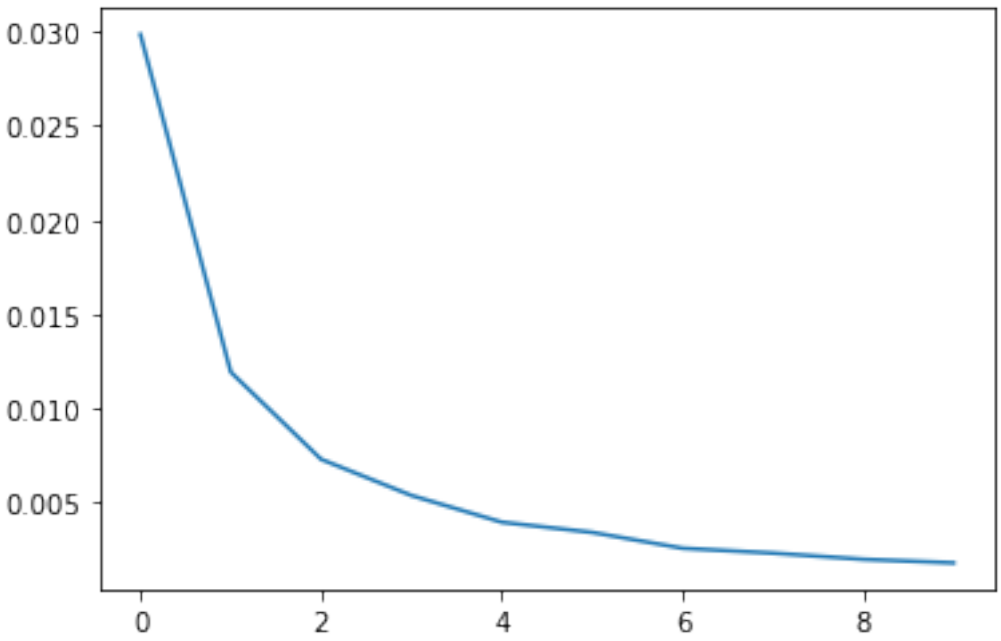
[10]:
```
# Calculate mean RSME of k portfolios of size n.
def portfolio_means(k, n):
    retval = 0
    for _ in range(k):
        selected = tickers.sample(n=n)
        cloned = mktcap_data[selected].copy()
        cloned['sum'] = cloned[selected].sum(axis=1)
        cloned[selected] = cloned[selected].div(cloned['sum'].values,axis=0)
        weighted_returns = pd.DataFrame()
        for each in selected:
            weighted_returns[each] = cloned[each] * return_data[each]
        weighted_returns['summed'] = weighted_returns.sum(axis=1)
        retval += ((weighted_returns['summed'] - return_data['avg']) ** 2).
↪mean() ** .5
    return retval/n
```

[11]:
```
portfolio_means(25, 10)
```

[11]: 0.028128808394006888

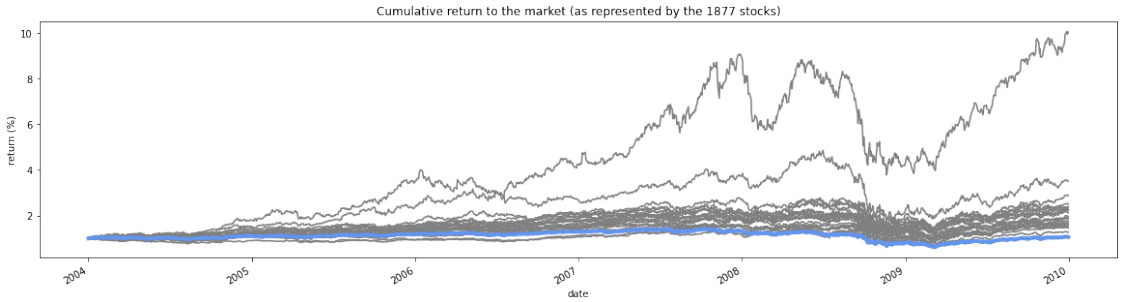
[12]:
```
# Repeat for n = 10, 20, etc.. and plot mean RMSE as a function of n.
rmse_n = pd.Series([portfolio_means(25, each) for each in range(10, 110, 10)])
rmse_n.plot()
```

[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2d58e75bd0>



[13]:
```
# Choose an n, and plot the cumulative returns to all 25 porfolios, and to the␣
 ↪market.
for _ in range(25):
    selected = tickers.sample(n=10)
    cloned = mktcap_data[selected].copy()
    cloned['sum'] = cloned[selected].sum(axis=1)
    cloned[selected] = cloned[selected].div(cloned['sum'].values,axis=0)
    weighted_returns = pd.DataFrame()
    for each in selected:
        weighted_returns[each] = cloned[each] * return_data[each]
    weighted_returns['summed'] = weighted_returns.sum(axis=1)
    cumulative_returns(weighted_returns['summed']).plot(ylabel='return (%)',␣
 ↪figsize=(20,5), title='Cumulative return to the market (as represented by␣
 ↪the 1877 stocks)', color="grey")

cumulative_returns(ff_ret['mkt_ret']).plot(lw=4, color="cornflowerblue")
```

[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2d781a3750>



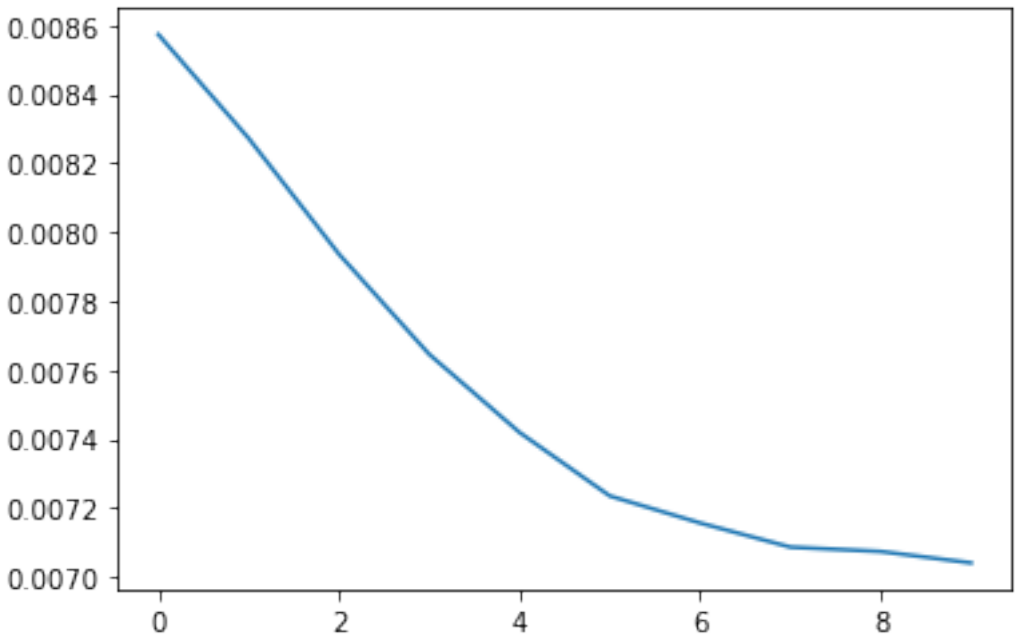
0.0.4 1D: Heuristic approach to choosing stocks.

[14]:
```
heap = []
first_row = mktcap_data.head(1)
for each in first_row:
    heapq.heappush(heap,(first_row[each].values[0], each))

def get_largest_mktcap(n):
    return [each[1] for each in heapq.nlargest(n, heap)]

get_largest_mktcap(10)
```

[14]:
```
['GE', 'MSFT', 'PFE', 'XOM', 'C', 'WMT', 'INTC', 'AIG', 'CSCO', 'IBM']
```

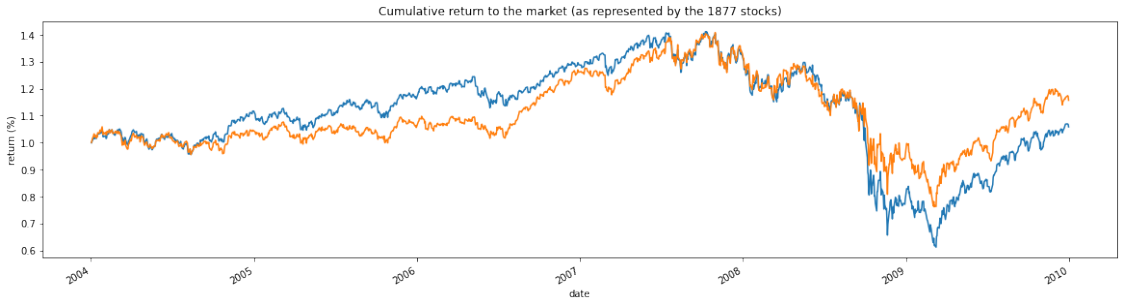
[15]:
```
# RMSE as function of number stocks in portfolio.
def rmse_by_cap(n):
    retval = 0
    selected = get_largest_mktcap(n)
    cloned = mktcap_data[selected].copy()
    cloned['sum'] = cloned[selected].sum(axis=1)
    cloned[selected] = cloned[selected].div(cloned['sum'].values,axis=0)
    weighted_returns = pd.DataFrame()
    for each in selected:
        weighted_returns[each] = cloned[each] * return_data[each]

    weighted_returns['summed'] = weighted_returns.sum(axis=1)
    return ((weighted_returns['summed'] - return_data['avg']) ** 2).mean() ** .5

rmse_n = pd.Series([rmse_by_cap(each) for each in range(10, 110, 10)])
rmse_n.plot()
```

[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2d587c9290>
```



[16]:
```
# Choose n (we chose 10), and plot the cumulative returns to your portfolio and␣
 ↪market.
selected = get_largest_mktcap(10)
cloned = mktcap_data[selected].copy()
cloned['sum'] = cloned[selected].sum(axis=1)
cloned[selected] = cloned[selected].div(cloned['sum'].values,axis=0)
weighted_returns = pd.DataFrame()

for each in selected:
    weighted_returns[each] = cloned[each] * return_data[each]

weighted_returns['summed'] = weighted_returns.sum(axis=1)
cumulative_returns(ff_ret['mkt_ret']).plot()
cumulative_returns(weighted_returns['summed']).plot(ylabel='return (%)',␣
 ↪figsize=(20,5), title='Cumulative return to the market (as represented by␣
 ↪the 1877 stocks)')
```

[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2d587c9e10>



0.0.5 2A: Pick portfolio of 100 securities and calculate rolling statistics.

[17]:
```
# Helper function from Anh-Tu
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_weights) ** 0.
↪5
```

[18]:
```
# Select hundred securities to be used throughout.
hundred_securities = tickers[:100]
return_data[hundred_securities].head()
```

[18]:

date	A	AA	AAI	AAON	AAP	AAPL	\
2004-01-02	-0.015048	-0.011842	0.030252	-0.024729	0.000246	-0.004212	
2004-01-05	0.026042	0.032756	0.008157	0.004754	-0.009089	0.041823	
2004-01-06	0.031472	-0.007478	0.062298	0.014196	0.017848	-0.003608	
2004-01-07	0.012795	-0.007534	0.022848	-0.005184	0.031417	0.022635	
2004-01-08	0.045675	0.012042	-0.067014	0.001042	-0.026446	0.034086	

date	ABAX	ABC	ABCB	ABCO	…	AMED	AMG	\
					…			
2004-01-02	-0.072416	-0.038290	0.011801	-0.004594	…	-0.010554	0.002443	
2004-01-05	-0.072706	0.033889	0.016574	0.003750	…	0.007267	0.035837	
2004-01-06	0.097044	-0.004120	0.006099	0.002299	…	0.018598	0.008718	
2004-01-07	0.062683	0.009712	0.012544	-0.001433	…	-0.018194	0.011524	
2004-01-08	0.006064	-0.004809	0.007706	-0.024117	…	0.018531	0.010443	

date	AMGN	AMKR	AMLN	AMMD	AMN	AMR	\
2004-01-02	0.009063	0.002205	0.006301	0.002292	0.006054	0.003861	
2004-01-05	0.000000	0.063256	-0.051878	0.009145	-0.005731	0.003846	
2004-01-06	0.003208	0.002587	0.015896	-0.021296	0.007205	0.049808	

```
2004-01-07  0.013269  0.018060  0.008961  0.016620  0.001431  0.018248
2004-01-08 -0.000316  0.030411  0.013806  0.018261 -0.000286 -0.020072

                AMRI      AMSC
date
2004-01-02  0.003997 -0.010822
2004-01-05 -0.039814  0.049599
2004-01-06  0.033863  0.050035
2004-01-07  0.010027  0.034414
2004-01-08 -0.004633  0.049264

[5 rows x 100 columns]
```

Rolling window: 504

[19]:
```
# Calculate sample covariance matrix (100 x 100) of returns to securities.
# Drop NAs and trail row (see one-day ahead portion).

cov_504 = return_data[hundred_securities].rolling(504).cov()
cov_504.dropna(inplace=True)
cov_504 = cov_504.drop(cov_504.tail(100).index)
cov_504.head()
```

[19]:

		A	AA	AAI	AAON	AAP	AAPL	\
date								
2005-12-30	A	0.000438	0.000085	0.000203	0.000117	0.000071	0.000140	
	AA	0.000085	0.000255	0.000123	0.000068	0.000062	0.000122	
	AAI	0.000203	0.000123	0.000910	0.000135	0.000131	0.000171	
	AAON	0.000117	0.000068	0.000135	0.000482	0.000065	0.000090	
	AAP	0.000071	0.000062	0.000131	0.000065	0.000316	0.000070	

		ABAX	ABC	ABCB	ABCO	…	AMED	\
date						…		
2005-12-30	A	0.000142	0.000058	0.000109	0.000071	…	0.000090	
	AA	0.000086	0.000053	0.000103	0.000056	…	0.000067	
	AAI	0.000280	0.000100	0.000219	0.000118	…	0.000065	
	AAON	0.000122	0.000040	0.000112	0.000053	…	0.000098	
	AAP	0.000099	0.000007	0.000089	0.000046	…	0.000053	

		AMG	AMGN	AMKR	AMLN	AMMD	AMN	\
date								
2005-12-30	A	0.000099	0.000073	0.000305	0.000144	0.000132	0.000099	
	AA	0.000076	0.000060	0.000163	0.000070	0.000094	0.000068	
	AAI	0.000131	0.000102	0.000325	0.000197	0.000201	0.000158	
	AAON	0.000061	0.000055	0.000150	0.000129	0.000100	0.000068	
	AAP	0.000055	0.000026	0.000087	0.000073	0.000069	0.000052	

```
                      AMR       AMRI      AMSC
date
2005-12-30 A     0.000189  0.000152  0.000228
           AA    0.000128  0.000106  0.000155
           AAI   0.000648  0.000197  0.000246
           AAON  0.000107  0.000150  0.000127
           AAP   0.000157  0.000082  0.000101

[5 rows x 100 columns]
```

[20]:
```
wgt_504 = mktcap_data[hundred_securities].copy()
wgt_504['sum'] = wgt_504[hundred_securities].sum(axis=1)
wgt_504[hundred_securities] = wgt_504[hundred_securities].div(wgt_504['sum'].
 ↪values,axis=0)
wgt_504.drop(wgt_504.tail(1).index, inplace=True)
```

[31]:
```
sd_504 = []
cov504_np = cov_504.to_numpy()
arr504_weight = wgt_504[hundred_securities].tail(1007).to_numpy()

for i in range(0, len(cov504_np), 100):
    idx = 0
    sd_504.append(sd_portfolio(cov504_np[i:i+100], arr504_weight[idx]))
    idx += 1

sd_504_np = np.array(sd_504)
sd_504_np
```

[31]:
```
array([0.00820984, 0.00824556, 0.00823351, …, 0.03161844, 0.03161853,
       0.03161824])
```

[34]:
```
dayahead_504 = []
ret_copy = return_data[hundred_securities].copy()
ret_copy.drop(ret_copy.tail(1).index, inplace=True)
dayahead_504_ret = ret_copy.tail(1007).to_numpy()
dayahead_504_wgt = wgt_504[hundred_securities].tail(1007).to_numpy()

for i in range(0, len(dayahead_504_wgt)):
    dayahead_504.append(np.multiply(dayahead_504_ret[i], dayahead_504_wgt[i]))

dayahead_504_ret_np = np.sum(np.array(dayahead_504), axis=1)
dayahead_504_ret_np
```

[34]:
```
array([-0.00618081,  0.01826034,  0.00564215, …,  0.00519065,
       -0.00414611,  0.00514476])
```

[37]:
```
std_outcomes = dayahead_505_ret_np / sd_504_np
std_504 = pd.DataFrame(std_outcomes)
std_504.index += 504
std_504.rename(columns={0: "Standardized Outcome"}, inplace=True)
std_504.head()
```

[37]:

	Standardized Outcome
504	-0.752854
505	2.214566
506	0.685266
507	-0.066878
508	1.336027

Rolling window: 252

[40]:
```
cov252 = return_data[hundred_securities].rolling(252).cov()
cov252.dropna(inplace=True)
cov252 = cov252.drop(cov252.tail(100).index)
cov252.head()

cloned = mktcap_data[hundred_securities].copy()
cloned['sum'] = cloned[hundred_securities].sum(axis=1)
cloned[hundred_securities] = cloned[hundred_securities].div(cloned['sum'].
 ↪values,axis=0)
cloned.drop(cloned.tail(1).index, inplace=True)
weights_252 = cloned

std_dev_252 = []
cov252_np = cov252.to_numpy()
arr252_weight = cloned[hundred_securities].tail(1259).to_numpy()

for i in range(0, len(cov252_np), 100):
    idx = 0
    std_dev_252.append(sd_portfolio(cov252_np[i:i+100], arr252_weight[idx]))
    idx += 1

std_dev_252_arr = np.array(std_dev_252)

dayahead_252 = []
ret_copy = return_data[hundred_securities].copy()
ret_copy.drop(ret_copy.tail(1).index, inplace=True)
dayahead_252_ret = ret_copy.tail(1259).to_numpy()
dayahead_252_wgt = weights_252[hundred_securities].tail(1259).to_numpy()
for i in range(0, len(dayahead_252_wgt)):
    dayahead_252.append(np.multiply(dayahead_252_ret[i], dayahead_252_wgt[i]))

dayahead_252_ret_arr = np.sum(np.array(dayahead_252), axis=1)
```

```
std_outcomes = dayahead_252_ret_arr / std_dev_252_arr
std_252 = pd.DataFrame(std_outcomes)
std_252.index += 252
std_252.rename(columns={0: "Standardized Outcome"}, inplace=True)
std_252.head()
```

[40]:

	Standardized Outcome
252	-0.087862
253	-0.757350
254	-1.347570
255	-0.294337
256	0.542214

Rolling window: 126

[45]:
```
cov_126 = return_data[hundred_securities].rolling(126).cov()
cov_126.dropna(inplace=True)
cov_126 = cov_126.drop(cov_126.tail(100).index)

cloned = mktcap_data[hundred_securities].copy()
cloned['sum'] = cloned[hundred_securities].sum(axis=1)
cloned[hundred_securities] = cloned[hundred_securities].div(cloned['sum'].
 ↪values,axis=0)
cloned.drop(cloned.tail(1).index, inplace=True)
weights_126 = cloned

std_dev_126 = []
cov126_np = cov_126.to_numpy()
arr126_weight = cloned[hundred_securities].tail(1385).to_numpy()

for i in range(0, len(cov126_np), 100):
    idx = 0
    std_dev_126.append(sd_portfolio(cov126_np[i:i+100], arr126_weight[idx]))
    idx += 1

std_dev_126_arr = np.array(std_dev_126)

dayahead_126 = []
ret_copy = return_data[hundred_securities].copy()
ret_copy.drop(ret_copy.tail(1).index, inplace=True)
dayahead_126_ret = ret_copy.tail(1385).to_numpy()
dayahead_126_wgt = weights_126[hundred_securities].tail(1385).to_numpy()
for i in range(0, len(dayahead_126_wgt)):
    dayahead_126.append(np.multiply(dayahead_126_ret[i], dayahead_126_wgt[i]))

dayahead_126_ret_arr = np.sum(np.array(dayahead_126), axis=1)
```

```
std_outcomes = dayahead_126_ret_arr / std_dev_126_arr
std_126 = pd.DataFrame(std_outcomes)
std_126.index += 126
std_126.rename(columns={0: "Standardized Outcome"}, inplace=True)
std_126.head()
```

[45]:

	Standardized Outcome
126	-0.431529
127	-1.488281
128	0.066728
129	-0.733234
130	-0.016902

Rolling window: 63

[46]:
```
covariance63 = return_data[hundred_securities].rolling(63).cov()
covariance63.dropna(inplace=True)
covariance63 = covariance63.drop(covariance63.tail(100).index)

cloned = mktcap_data[hundred_securities].copy()
cloned['sum'] = cloned[hundred_securities].sum(axis=1)
cloned[hundred_securities] = cloned[hundred_securities].div(cloned['sum'].
 ↪values,axis=0)
cloned.drop(cloned.tail(1).index, inplace=True)
weights_63 = cloned

std_dev_63 = []
cov63_np = covariance63.to_numpy()
arr63_weight = cloned[hundred_securities].tail(1448).to_numpy()

for i in range(0, len(cov63_np), 100):
    idx = 0
    std_dev_63.append(sd_portfolio(cov63_np[i:i+100], arr63_weight[idx]))
    idx += 1

std_dev_63_arr = np.array(std_dev_63)

dayahead_63 = []
ret_copy = return_data[hundred_securities].copy()
ret_copy.drop(ret_copy.tail(1).index, inplace=True)
dayahead_63_ret = ret_copy.tail(1448).to_numpy()
dayahead_63_wgt = weights_63[hundred_securities].tail(1448).to_numpy()
for i in range(0, len(dayahead_63_wgt)):
    dayahead_63.append(np.multiply(dayahead_63_ret[i], dayahead_63_wgt[i]))

dayahead_63_ret_arr = np.sum(np.array(dayahead_63), axis=1)
```

```
std_outcomes = dayahead_63_ret_arr / std_dev_63_arr
std_63 = pd.DataFrame(std_outcomes)
std_63.index += 63
std_63.rename(columns={0: "Standardized Outcome"}, inplace=True)
std_63.head()
```

[46]:

	Standardized Outcome
63	1.687199
64	1.914711
65	1.287970
66	-0.546151
67	-0.526226

[48]:
```
# Various standardized outcomes from different rolling window sizes.
print("Rolling window 504:", np.std(std_504))
print("Rolling window 252:", np.std(std_252))
print("Rolling window 126:", np.std(std_126))
print("Rolling window 63:", np.std(std_63))
```

```
Rolling window 504: Standardized Outcome    1.157416
dtype: float64
Rolling window 252: Standardized Outcome    1.025599
dtype: float64
Rolling window 126: Standardized Outcome    0.952773
dtype: float64
Rolling window 63: Standardized Outcome    0.923049
dtype: float64
```

0.0.6 3. CAPM Model

[64]:
```
stock_returns = return_data[hundred_securities]
mkt_caps = mktcap_data[hundred_securities].copy()
mkt_caps['sum'] = mkt_caps[hundred_securities].sum(axis=1)
mkt_caps[hundred_securities] = mkt_caps[hundred_securities].div(mkt_caps['sum'].
 ↪values,axis=0)

equity_prem = stock_returns.sub(ff_ret["rf_rate"], axis=0)
mkt_prem = pd.DataFrame(ff_ret['mkt_ret'] - ff_ret['rf_rate'])
mkt_prem.columns = ["mkt_prem"]
mkt_caps[hundred_securities].head()
```

[64]:

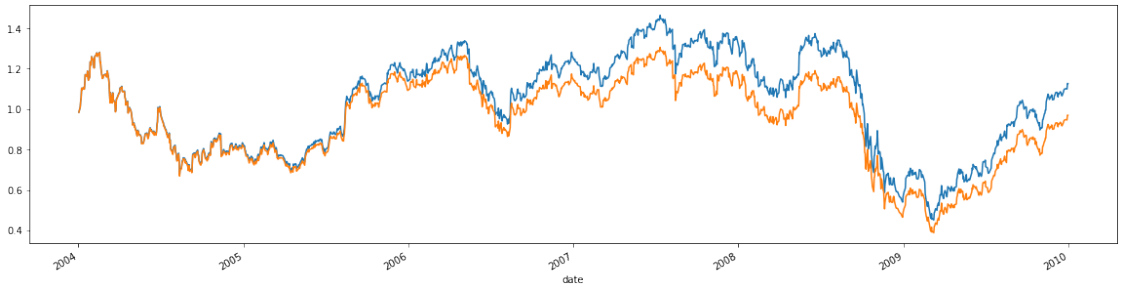
	A	AA	AAI	AAON	AAP	AAPL	\
date							
2004-01-02	0.018568	0.043998	0.001397	0.000321	0.004069	0.010625	
2004-01-05	0.018809	0.044861	0.001390	0.000318	0.003980	0.010929	

```
2004-01-06  0.019375  0.044465  0.001475  0.000322  0.004046  0.010875
2004-01-07  0.019478  0.043805  0.001497  0.000318  0.004142  0.011039
2004-01-08  0.020277  0.044135  0.001391  0.000317  0.004015  0.011364

                ABAX       ABC       ABCB       ABCO  …      AMED       AMG  \
date                                                    …
2004-01-02  0.000441  0.008199  0.000216  0.000741  …  0.000242  0.002014
2004-01-05  0.000403  0.008369  0.000217  0.000734  …  0.000240  0.002060
2004-01-06  0.000442  0.008323  0.000218  0.000735  …  0.000245  0.002075
2004-01-07  0.000466  0.008342  0.000219  0.000728  …  0.000238  0.002084
2004-01-08  0.000467  0.008265  0.000219  0.000708  …  0.000242  0.002096

                AMGN      AMKR      AMLN      AMMD       AMN       AMR  \
date
2004-01-02  0.108375  0.004296  0.002847  0.000981  0.000382  0.002805
2004-01-05  0.106997  0.004509  0.002665  0.000978  0.000375  0.002780
2004-01-06  0.107192  0.004515  0.002703  0.000955  0.000377  0.002914
2004-01-07  0.107815  0.004563  0.002707  0.000964  0.000375  0.002946
2004-01-08  0.107301  0.004680  0.002733  0.000977  0.000373  0.002874

                AMRI      AMSC
date
2004-01-02  0.000645  0.000505
2004-01-05  0.000612  0.000523
2004-01-06  0.000631  0.000549
2004-01-07  0.000633  0.000564
2004-01-08  0.000627  0.000589

[5 rows x 100 columns]
```

[66]:
```
cumulative_returns(stock_returns["A"]).plot()
cumulative_returns(equity_prem["A"]).plot(figsize=(20,5))
stock_returns.head()
```

[66]:
```
                   A         AA        AAI       AAON       AAP      AAPL  \
date
2004-01-02 -0.015048 -0.011842  0.030252 -0.024729  0.000246 -0.004212
2004-01-05  0.026042  0.032756  0.008157  0.004754 -0.009089  0.041823
2004-01-06  0.031472 -0.007478  0.062298  0.014196  0.017848 -0.003608
2004-01-07  0.012795 -0.007534  0.022848 -0.005184  0.031417  0.022635
2004-01-08  0.045675  0.012042 -0.067014  0.001042 -0.026446  0.034086

                ABAX       ABC       ABCB       ABCO  …      AMED       AMG  \
date                                                    …
2004-01-02 -0.072416 -0.038290  0.011801 -0.004594  … -0.010554  0.002443
2004-01-05 -0.072706  0.033889  0.016574  0.003750  …  0.007267  0.035837
2004-01-06  0.097044 -0.004120  0.006099  0.002299  …  0.018598  0.008718
```

```
2004-01-07  0.062683  0.009712  0.012544 -0.001433  … -0.018194  0.011524
2004-01-08  0.006064 -0.004809  0.007706 -0.024117  …  0.018531  0.010443

                AMGN      AMKR      AMLN      AMMD      AMN       AMR  \
date
2004-01-02  0.009063  0.002205  0.006301  0.002292  0.006054  0.003861
2004-01-05  0.000000  0.063256 -0.051878  0.009145 -0.005731  0.003846
2004-01-06  0.003208  0.002587  0.015896 -0.021296  0.007205  0.049808
2004-01-07  0.013269  0.018060  0.008961  0.016620  0.001431  0.018248
2004-01-08 -0.000316  0.030411  0.013806  0.018261 -0.000286 -0.020072

                AMRI      AMSC
date
2004-01-02  0.003997 -0.010822
2004-01-05 -0.039814  0.049599
2004-01-06  0.033863  0.050035
2004-01-07  0.010027  0.034414
2004-01-08 -0.004633  0.049264

[5 rows x 100 columns]
```



[68]:
```
beta_frame = pd.DataFrame()
for each in hundred_securities:
    endog = equity_prem[each]
    exog = sm.add_constant(mkt_prem)
    rols = RollingOLS(endog, exog, window=504)
    rres = rols.fit()
    params = rres.params
    params.dropna(inplace=True)
    beta_frame[each] = pd.Series(params['mkt_prem'])

variances = stock_returns.rolling(window=504).std()
variances.dropna(inplace=True)
variances.drop(variances.tail(1).index, inplace=True)
variances_s = variances**2
```

[69]:
```
variances.head()
```

[69]:

date	A	AA	AAI	AAON	AAP	AAPL	\
2005-12-30	0.020934	0.015983	0.030162	0.021961	0.017767	0.024961	
2006-01-03	0.020924	0.015983	0.030144	0.021946	0.017767	0.025009	
2006-01-04	0.020893	0.015917	0.030164	0.021946	0.017763	0.024953	
2006-01-05	0.020879	0.015918	0.030053	0.021938	0.017748	0.024956	
2006-01-06	0.020873	0.015916	0.030051	0.021938	0.017697	0.024961	

date	ABAX	ABC	ABCB	ABCO	…	AMED	AMG	\
					…			
2005-12-30	0.033430	0.014042	0.019016	0.017888	…	0.030033	0.012280	
2006-01-03	0.033277	0.013961	0.019013	0.017887	…	0.030033	0.012335	
2006-01-04	0.033234	0.013888	0.019002	0.017902	…	0.030068	0.012240	
2006-01-05	0.032957	0.013898	0.019008	0.017936	…	0.030101	0.012241	
2006-01-06	0.032840	0.013903	0.019002	0.017944	…	0.030101	0.012248	

date	AMGN	AMKR	AMLN	AMMD	AMN	AMR	\
2005-12-30	0.015255	0.042950	0.028721	0.021874	0.019898	0.033482	
2006-01-03	0.015273	0.042951	0.028720	0.021879	0.019901	0.033492	
2006-01-04	0.015275	0.042854	0.028621	0.021898	0.019974	0.033506	
2006-01-05	0.015289	0.042900	0.028614	0.021882	0.019976	0.033438	
2006-01-06	0.015285	0.042892	0.028665	0.022927	0.020037	0.033430	

date	AMRI	AMSC
2005-12-30	0.030172	0.033548
2006-01-03	0.030172	0.033702
2006-01-04	0.030121	0.033725
2006-01-05	0.030084	0.033652
2006-01-06	0.030083	0.033616

[5 rows x 100 columns]

[136]:
```
var_market = ff_ret['mkt_ret'].rolling(window=504).var()

var_market.dropna(inplace=True)
var_market.drop(var_market.tail(1).index, inplace=True)
var_market.head()
```

[136]:
```
date
2005-12-30    0.000046
2006-01-03    0.000047
2006-01-04    0.000046
2006-01-05    0.000046
```

```
2006-01-06    0.000046
Name: mkt_ret, dtype: float64
```

[111]:
```
beta_frame.drop(beta_frame.tail(1).index, inplace=True)
beta_frame.head()
```

[111]:

date	A	AA	AAI	AAON	AAP	AAPL	\
2005-12-30	1.585701	1.399423	2.199838	1.140875	0.979726	1.564999	
2006-01-03	1.572052	1.391717	2.189794	1.139188	0.969457	1.572019	
2006-01-04	1.567372	1.383090	2.202483	1.141956	0.980790	1.560432	
2006-01-05	1.564328	1.383358	2.199246	1.140902	0.979563	1.561555	
2006-01-06	1.559648	1.378236	2.197591	1.134948	0.972259	1.562901	

date	ABAX	ABC	ABCB	ABCO	…	AMED	AMG	\
2005-12-30	1.804604	0.771034	1.563704	0.864623	…	0.923322	1.163837	
2006-01-03	1.790731	0.774352	1.542474	0.857462	…	0.904026	1.169035	
2006-01-04	1.850365	0.759819	1.543672	0.863799	…	0.913005	1.158403	
2006-01-05	1.844119	0.760751	1.542956	0.864748	…	0.910560	1.158310	
2006-01-06	1.832636	0.752554	1.537134	0.857361	…	0.917932	1.158786	

date	AMGN	AMKR	AMLN	AMMD	AMN	AMR	\
2005-12-30	1.069651	2.303654	1.508272	1.441555	1.357317	2.180845	
2006-01-03	1.071085	2.275491	1.492776	1.433408	1.336822	2.144903	
2006-01-04	1.075662	2.253836	1.527962	1.441190	1.355008	2.159020	
2006-01-05	1.076163	2.251941	1.526961	1.443354	1.354293	2.155618	
2006-01-06	1.067407	2.239527	1.536362	1.497452	1.363356	2.148651	

date	AMRI	AMSC
2005-12-30	1.602392	2.273857
2006-01-03	1.586640	2.298412
2006-01-04	1.616545	2.296423
2006-01-05	1.614214	2.293455
2006-01-06	1.611222	2.282349

[5 rows x 100 columns]

[112]:
```
# Helper function from Anh-Tu
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)
```

[120]:
```
residual_var = pd.DataFrame()
for each in hundred_securities:
```

```
    endog = equity_prem[each]
    exog = sm.add_constant(mkt_prem)
    rols = RollingOLS(endog, exog, window=504)
    rres = rols.fit()
    resids = rres.mse_resid
    resids.dropna(inplace=True)
    residual_var[each] = resids

residual_var.drop(residual_var.tail(1).index, inplace=True)

residual_var["A"].to_numpy()
residual_var.head()
```

[120]:

date	A	AA	AAI	AAON	AAP	AAPL	\
2005-12-30	0.000323	0.000166	0.000688	0.000423	0.000272	0.000511	
2006-01-03	0.000323	0.000166	0.000687	0.000422	0.000272	0.000512	
2006-01-04	0.000323	0.000165	0.000687	0.000422	0.000272	0.000511	
2006-01-05	0.000323	0.000165	0.000681	0.000422	0.000271	0.000511	
2006-01-06	0.000323	0.000165	0.000680	0.000422	0.000270	0.000511	

date	ABAX	ABC	ABCB	ABCO	…	AMED	AMG	\
2005-12-30	0.000970	0.000170	0.000250	0.000286	…	0.000865	0.000089	
2006-01-03	0.000960	0.000167	0.000251	0.000286	…	0.000866	0.000089	
2006-01-04	0.000948	0.000166	0.000251	0.000287	…	0.000867	0.000088	
2006-01-05	0.000931	0.000167	0.000252	0.000288	…	0.000869	0.000088	
2006-01-06	0.000924	0.000167	0.000252	0.000288	…	0.000869	0.000088	

date	AMGN	AMKR	AMLN	AMMD	AMN	AMR	\
2005-12-30	0.000180	0.001603	0.000721	0.000384	0.000312	0.000904	
2006-01-03	0.000180	0.001607	0.000722	0.000384	0.000314	0.000909	
2006-01-04	0.000180	0.001604	0.000712	0.000384	0.000315	0.000909	
2006-01-05	0.000180	0.001609	0.000712	0.000383	0.000315	0.000905	
2006-01-06	0.000181	0.001610	0.000713	0.000422	0.000316	0.000905	

date	AMRI	AMSC
2005-12-30	0.000794	0.000889
2006-01-03	0.000795	0.000892
2006-01-04	0.000788	0.000895
2006-01-05	0.000786	0.000891
2006-01-06	0.000786	0.000890

[5 rows x 100 columns]

[138]:
```
# Data to be run through variance equation.
w = mkt_caps[hundred_securities].copy()
w.drop(w.tail(1).index, inplace=True)
w = w.tail(1007)
B = beta_frame.to_numpy()
delta = residual_var
```

[129]:
```
var_market
```

[129]:
```
array([4.60195227e-05, 4.65171063e-05, 4.62866136e-05, …,
       4.84033028e-04, 4.84025332e-04, 4.83941549e-04])
```

[169]:
```
var_port_504 = []
for i in range(len(w)-1):
    diag = np.diag(delta.iloc[i])
    res = var_portfolio(w.iloc[i], beta_frame.iloc[i], var_market.iloc[i], diag)
    var_port_504.append(res)

var_port_504 = np.array(var_port_504)
var_port_504
```

[169]:
```
array([6.64230302e-05, 6.73470699e-05, 6.70732969e-05, …,
       4.48227879e-04, 4.47462822e-04, 4.46292944e-04])
```

[170]:
```
var_port_504_rt = np.sqrt(var_port_504)
var_port_504_rt
```

[170]:
```
array([0.00815003, 0.00820653, 0.00818983, …, 0.02117139, 0.02115332,
       0.02112565])
```

[171]:
```
ahead_504 = []
ret = stock_returns.tail(1007)
wgt = w.tail(1007)
for i in range(len(w)-1):
    ahead_504.append(np.multiply(ret.iloc[i], w.iloc[i]))

arr = np.sum(np.array(ahead_504), axis = 1)
arr
```

[171]:
```
array([ 0.01805583,  0.00549331, -0.00071404, …,  0.00509209,
       -0.00421917,  0.0050682 ])
```

[184]:
```
stdout = var_port_504/arr
stdout = pd.DataFrame(stdout)
stdout.index += 504
stdout.rename(columns={0: "Standard Outcome"}, inplace=True)
stdout
```

[184]:
```
      Standard Outcome
504           0.004573
505           0.007176
506          -0.016053
507          -0.016641
508           0.051053
…                …
1946          0.026113
1947          0.009585
1948          0.022787
1949         -0.025962
1950          0.021577

[1447 rows x 1 columns]
```

0.0.7 Rolling window: 252

[185]:
```
beta_frame = pd.DataFrame()
for each in hundred_securities:
    endog = equity_prem[each]
    exog = sm.add_constant(mkt_prem)
    rols = RollingOLS(endog, exog, window=252)
    rres = rols.fit()
    params = rres.params
    params.dropna(inplace=True)
    beta_frame[each] = pd.Series(params['mkt_prem'])

variances = stock_returns.rolling(window=252).std()
variances.dropna(inplace=True)
variances.drop(variances.tail(1).index, inplace=True)
variances_s = variances**2

var_market = ff_ret['mkt_ret'].rolling(window=252).var()

var_market.dropna(inplace=True)
var_market.drop(var_market.tail(1).index, inplace=True)
var_market.head()
beta_frame.drop(beta_frame.tail(1).index, inplace=True)

residual_var = pd.DataFrame()
for each in hundred_securities:
    endog = equity_prem[each]
    exog = sm.add_constant(mkt_prem)
    rols = RollingOLS(endog, exog, window=252)
    rres = rols.fit()
    resids = rres.mse_resid
```

```
    resids.dropna(inplace=True)
    residual_var[each] = resids

residual_var.drop(residual_var.tail(1).index, inplace=True)

residual_var["A"].to_numpy()
residual_var.head()

# Data to be run through variance equation.
w = mkt_caps[hundred_securities].copy()
w.drop(w.tail(1).index, inplace=True)
w = w.tail(1259)
B = beta_frame.to_numpy()
delta = residual_var

var_port_504 = []
for i in range(len(w)-1):
    diag = np.diag(delta.iloc[i])
    res = var_portfolio(w.iloc[i], beta_frame.iloc[i], var_market.iloc[i], diag)
    var_port_504.append(res)

var_port_504 = np.array(var_port_504)
var_port_504

var_port_504_rt = np.sqrt(var_port_504)


ahead_504 = []
ret = stock_returns.tail(1259)
wgt = w.tail(1259)
for i in range(len(w)-1):
    ahead_504.append(np.multiply(ret.iloc[i], w.iloc[i]))

arr = np.sum(np.array(ahead_504), axis = 1)
arr

stdout252 = var_port_504/arr
stdout252 = pd.DataFrame(stdout252)
stdout252.index += 252
stdout252.rename(columns={0: "Standard Outcome"}, inplace=True)
stdout252
```

[185]:

	Standard Outcome
252	-0.010819
253	-0.006091
254	-0.026079
255	0.015907

```
256           0.013720
…                …
1505          0.062987
1506          0.023534
1507          0.055237
1508         -0.066441
1509          0.055182

[1258 rows x 1 columns]
```

0.0.8 Rolling window: 126

[186]:
```
beta_frame = pd.DataFrame()
for each in hundred_securities:
    endog = equity_prem[each]
    exog = sm.add_constant(mkt_prem)
    rols = RollingOLS(endog, exog, window=126)
    rres = rols.fit()
    params = rres.params
    params.dropna(inplace=True)
    beta_frame[each] = pd.Series(params['mkt_prem'])

variances = stock_returns.rolling(window=126).std()
variances.dropna(inplace=True)
variances.drop(variances.tail(1).index, inplace=True)
variances_s = variances**2

var_market = ff_ret['mkt_ret'].rolling(window=126).var()

var_market.dropna(inplace=True)
var_market.drop(var_market.tail(1).index, inplace=True)
var_market.head()
beta_frame.drop(beta_frame.tail(1).index, inplace=True)

residual_var = pd.DataFrame()
for each in hundred_securities:
    endog = equity_prem[each]
    exog = sm.add_constant(mkt_prem)
    rols = RollingOLS(endog, exog, window=126)
    rres = rols.fit()
    resids = rres.mse_resid
    resids.dropna(inplace=True)
    residual_var[each] = resids

residual_var.drop(residual_var.tail(1).index, inplace=True)
```

```
residual_var["A"].to_numpy()
residual_var.head()

# Data to be run through variance equation.
w = mkt_caps[hundred_securities].copy()
w.drop(w.tail(1).index, inplace=True)
w = w.tail(1385)
B = beta_frame.to_numpy()
delta = residual_var

var_port_504 = []
for i in range(len(w)-1):
    diag = np.diag(delta.iloc[i])
    res = var_portfolio(w.iloc[i], beta_frame.iloc[i], var_market.iloc[i], diag)
    var_port_504.append(res)

var_port_504 = np.array(var_port_504)
var_port_504

var_port_504_rt = np.sqrt(var_port_504)


ahead_504 = []
ret = stock_returns.tail(1385)
wgt = w.tail(1385)
for i in range(len(w)-1):
    ahead_504.append(np.multiply(ret.iloc[i], w.iloc[i]))

arr = np.sum(np.array(ahead_504), axis = 1)
arr

stdout126 = var_port_504/arr
stdout126 = pd.DataFrame(stdout126)
stdout126.index += 126
stdout126.rename(columns={0: "Standard Outcome"}, inplace=True)
stdout126
```

[186]:

	Standard Outcome
126	-0.005443
127	0.150221
128	-0.010925
129	-0.267845
130	-0.591623
…	…
1505	0.025006
1506	0.009071
1507	0.021429

```
1508         -0.025813
1509          0.021377

[1384 rows x 1 columns]
```

0.0.9 Rolling window: 63

[187]:
```
beta_frame = pd.DataFrame()
for each in hundred_securities:
    endog = equity_prem[each]
    exog = sm.add_constant(mkt_prem)
    rols = RollingOLS(endog, exog, window=63)
    rres = rols.fit()
    params = rres.params
    params.dropna(inplace=True)
    beta_frame[each] = pd.Series(params['mkt_prem'])

variances = stock_returns.rolling(window=63).std()
variances.dropna(inplace=True)
variances.drop(variances.tail(1).index, inplace=True)
variances_s = variances**2

var_market = ff_ret['mkt_ret'].rolling(window=63).var()

var_market.dropna(inplace=True)
var_market.drop(var_market.tail(1).index, inplace=True)
var_market.head()
beta_frame.drop(beta_frame.tail(1).index, inplace=True)

residual_var = pd.DataFrame()
for each in hundred_securities:
    endog = equity_prem[each]
    exog = sm.add_constant(mkt_prem)
    rols = RollingOLS(endog, exog, window=63)
    rres = rols.fit()
    resids = rres.mse_resid
    resids.dropna(inplace=True)
    residual_var[each] = resids

residual_var.drop(residual_var.tail(1).index, inplace=True)

residual_var["A"].to_numpy()
residual_var.head()

# Data to be run through variance equation.
w = mkt_caps[hundred_securities].copy()
```

```
w.drop(w.tail(1).index, inplace=True)
w = w.tail(1448)
B = beta_frame.to_numpy()
delta = residual_var

var_port_504 = []
for i in range(len(w)-1):
    diag = np.diag(delta.iloc[i])
    res = var_portfolio(w.iloc[i], beta_frame.iloc[i], var_market.iloc[i], diag)
    var_port_504.append(res)

var_port_504 = np.array(var_port_504)
var_port_504

var_port_504_rt = np.sqrt(var_port_504)


ahead_504 = []
ret = stock_returns.tail(1448)
wgt = w.tail(1448)
for i in range(len(w)-1):
    ahead_504.append(np.multiply(ret.iloc[i], w.iloc[i]))

arr = np.sum(np.array(ahead_504), axis = 1)
arr

stdout63 = var_port_504/arr
stdout63 = pd.DataFrame(stdout63)
stdout63.index += 63
stdout63.rename(columns={0: "Standard Outcome"}, inplace=True)
stdout63
```

[187]:

	Standard Outcome
63	0.004573
64	0.007176
65	-0.016053
66	-0.016641
67	0.051053
…	…
1505	0.026113
1506	0.009585
1507	0.022787
1508	-0.025962
1509	0.021577

[1447 rows x 1 columns]

[190]:
```
print("window_size: 504, ", 1/np.std(stdout))
print("window_size: 252, ", 1/np.std(stdout252))
print("window_size: 126, ", 1/np.std(stdout126))
print("window_size: 63, ", 1/np.std(stdout63))
```

```
window_size: 504,  Standard Outcome    1.38355
dtype: float64
window_size: 252,  Standard Outcome    1.423428
dtype: float64
window_size: 126,  Standard Outcome    1.559759
dtype: float64
window_size: 63,  Standard Outcome    1.38355
dtype: float64
```

[ ]: