Zhang_James_HW_2

March 9, 2024

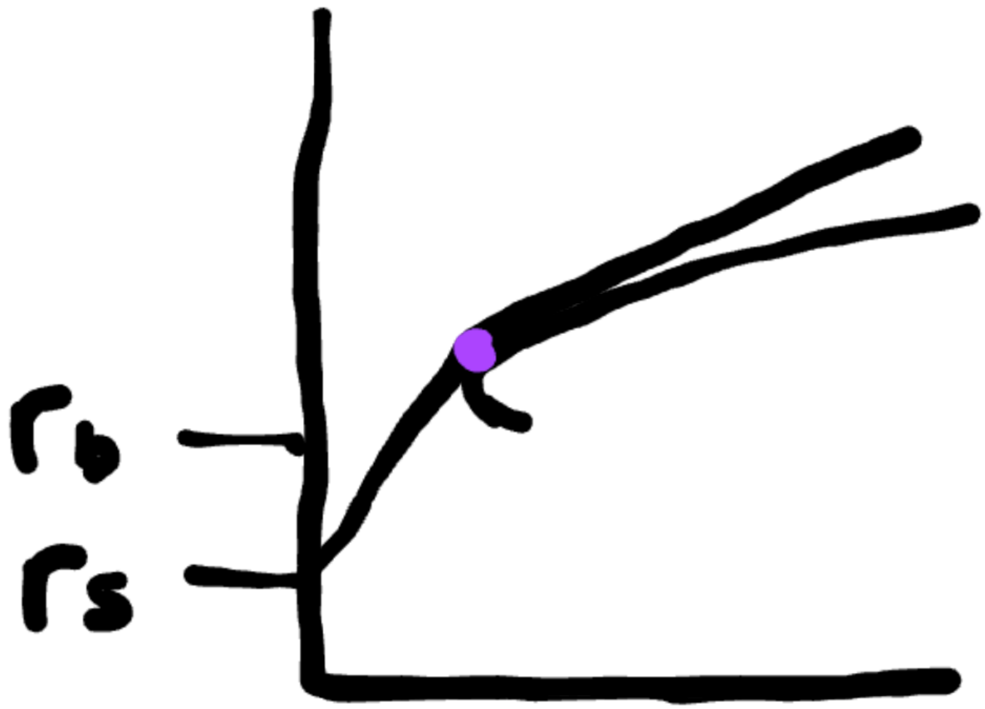
1 BUFN402: Homework 2

1.1 By James Zhang, 118843940

[3]:
```
import pandas as pd
import numpy as np
from scipy.optimize import minimize
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

1.1.1 1a

CAPM assumes that all investors face the same investment opportunities, and they borrow at the same equilibrium risk-free rate. Recall that the risk-free savings rate r_s is the rate of return of an investment with no risk of financial loss and the risk-free borrowing rate r_b is the rate at which someone can borrow funds. Assuming this is not true, and that the risk-free saving rate r_s and risk-free borrowing rate are different such that $r_s < r_b$, then the mean variance frontier will be two line segments, which pivots at the tangency portfolio. Below the purple dot, the investor buys the risk free savings rate at r_s and after the point sells to remain leveraged at the r_b rate.



1.1.2 1b

This table does not satisfy the CAPM's results. An important result of the CAPM is that since every trader is a mean-variance optimizer. Even if traders have different risk-tolerances, they will hold the same risky tangent portfolio T with maximum Sharpe Ratio, just with different weight to the tangent portfolio's weights. The different risk aversion (tolerances) simply adjust the weight of the risk-free asset and the weight of the tangent portfolio. However, the ratio of bonds to stocks for all of the investors should be the same. Since they aren't, not all investors are holding the tangent portfolio, so this defies the CAPM.

1.1.3 2a

[4]:
```
rf = 0.01
# Convert percentages to decimals by dividing by 100
df = pd.read_csv("ind30_m_vw_rets.csv").drop(columns=["Unnamed: 0"]) / 100
# Remove weird spaces from column names
df = df.rename(columns={col: col.replace(' ', '') for col in df.columns})
# Part a) Multiply df by 12 to annualize returns and covariances
df *= 12
df.head()
```

[4]:
```
     Food    Beer   Smoke   Games   Books   Hshld   Clths    Hlth   Chems  \
0  0.0672 -0.6228  0.1548  0.3516  1.3164 -0.0576  0.9696  0.2124  0.9768
```

```
1  0.3108  3.2436  0.7800  0.0660  1.2012 -0.4296 -0.3012  0.5100  0.6600
2  0.1392  0.4824  0.1512  0.7896 -0.1188  0.0876 -0.0612  0.0828  0.6396
3 -0.3672 -0.3972  0.1272 -0.5712  1.1364 -0.5616  0.0144 -0.0684 -0.5712
4  0.7620  0.8748  0.5460  0.1992 -0.6960 -0.0648  0.2244  0.6504  0.6240

    Txtls  …   Telcm   Servs   BusEq   Paper   Trans   Whlsl   Rtail  \
0  0.0468  …  0.0996  1.1064  0.2472  0.9240  0.2316 -2.8548  0.0084
1  0.9768  …  0.2604  0.2424  0.5268 -0.2856  0.5856  0.6468 -0.0900
2  0.2772  …  0.2892  0.2700  0.0228 -0.6648  0.0060 -0.9444  0.0300
3  0.1200  … -0.0132 -0.2400 -0.1308 -0.6096 -0.3168 -1.8456 -0.2640
4  0.3732  …  0.1956  0.4524  0.4368  0.4608  0.1920  0.5604  0.7824

    Meals     Fin   Other
0  0.2244  0.0444  0.6240
1 -0.0156  0.5352  0.8112
2 -0.0672 -0.1476 -0.4632
3 -0.4932 -0.6192 -1.0188
4  0.5196  0.2688  0.4800

[5 rows x 30 columns]
```

1.1.4 2b

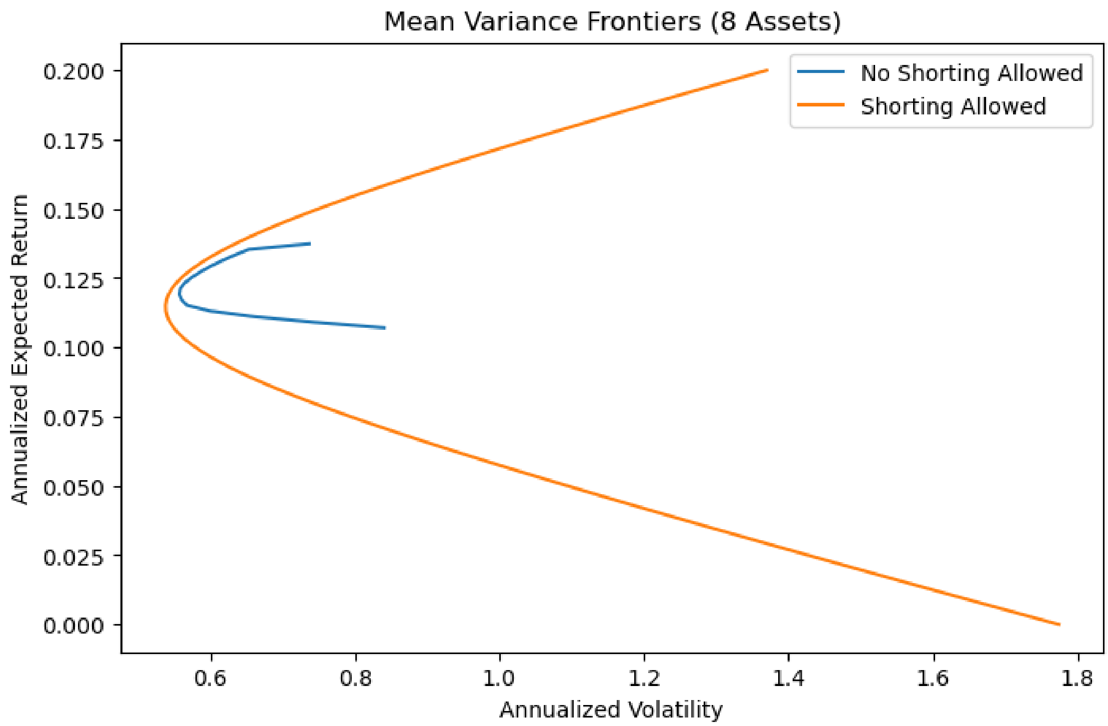
Consider the case of N risky assets. Let us define $w : N \times 1$ is a column vector of portfolio weights, $\mu : N \times 1$ is a column vector of risky asset expected returns, $\sigma : N \times 1$ is a column vector of risky asset standard deviations, and $\Sigma : N \times N$ is the covariance matrix.

$$E(R_p) = w^T \mu$$

$$Var(R_p) = w^T \Sigma w$$

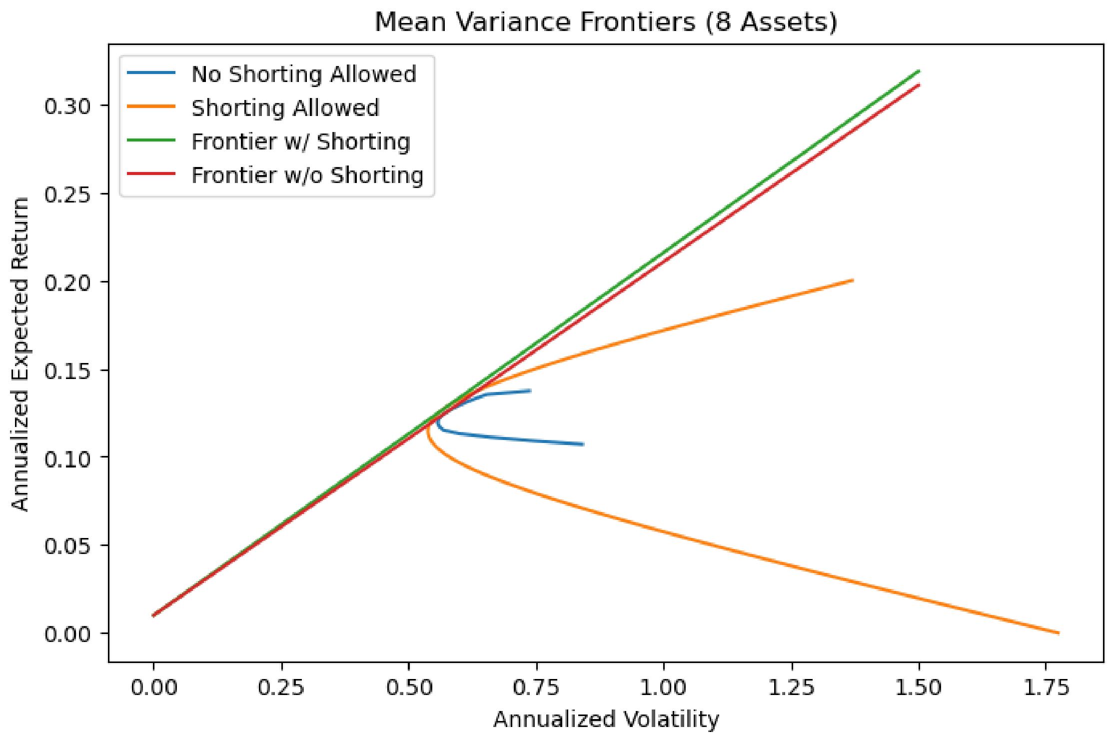
[5]:
```python
# Cutting df down to contain our risky assets
df_8 = df[["Smoke", "Fin", "Games", "Coal", "Food", "Autos", "Books", "ElcEq"]]
# Algorithm: fix desired expected return and use scipy.optimize to minimize portfolio variance
expected_returns = np.linspace(0, 0.2, 100)
no_shorting_expected_returns = []
no_shorting_volatilities, yes_shorting_volatilities = [], []
N = 8
mu, Sigma = np.array(df_8.mean()), np.array(df_8.cov())

def objective_func(weights, cov_matrix):
    portfolio_volatility = np.sqrt(weights.T @ cov_matrix @ weights)
    return portfolio_volatility


for i, target_return in enumerate(expected_returns):
```

```
    # Define constraints
    sum_to_one_constraint = {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}
    target_return_constraint = {'type': 'eq', 'fun': lambda weights: np.dot(weights, mu) - target_return}
    cons = [target_return_constraint, sum_to_one_constraint]

    # Not allowing shorting case
    w0 = np.ones(N) / N
    res = minimize(objective_func, x0=w0, args=(Sigma,), constraints=cons, bounds=[(0, None)] * N, tol=1e-6)
    if res.success:
        no_shorting_expected_returns.append(target_return)
        no_shorting_volatilities.append(res.fun)

    # Allowing shorting case
    w0 = np.random.random(N)
    w0 /= np.sum(w0)
    yes_shorting_min_vol = minimize(objective_func, x0=w0, args=(Sigma,), constraints=cons, tol=1e-6).fun
    yes_shorting_volatilities.append(yes_shorting_min_vol)

no_shorting_expected_returns = np.array(no_shorting_expected_returns)
no_shorting_volatilities = np.array(no_shorting_volatilities)
yes_shorting_volatilities = np.array(yes_shorting_volatilities)

fig, ax = plt.subplots(figsize=(8, 5))
plt.plot(no_shorting_volatilities, no_shorting_expected_returns, label="No Shorting Allowed")
plt.plot(yes_shorting_volatilities, expected_returns, label="Shorting Allowed")
ax.set_title("Mean Variance Frontiers (8 Assets)")
ax.set_xlabel("Annualized Volatility")
ax.set_ylabel("Annualized Expected Return")
ax.legend()
fig.show()
```



1.1.5 2c

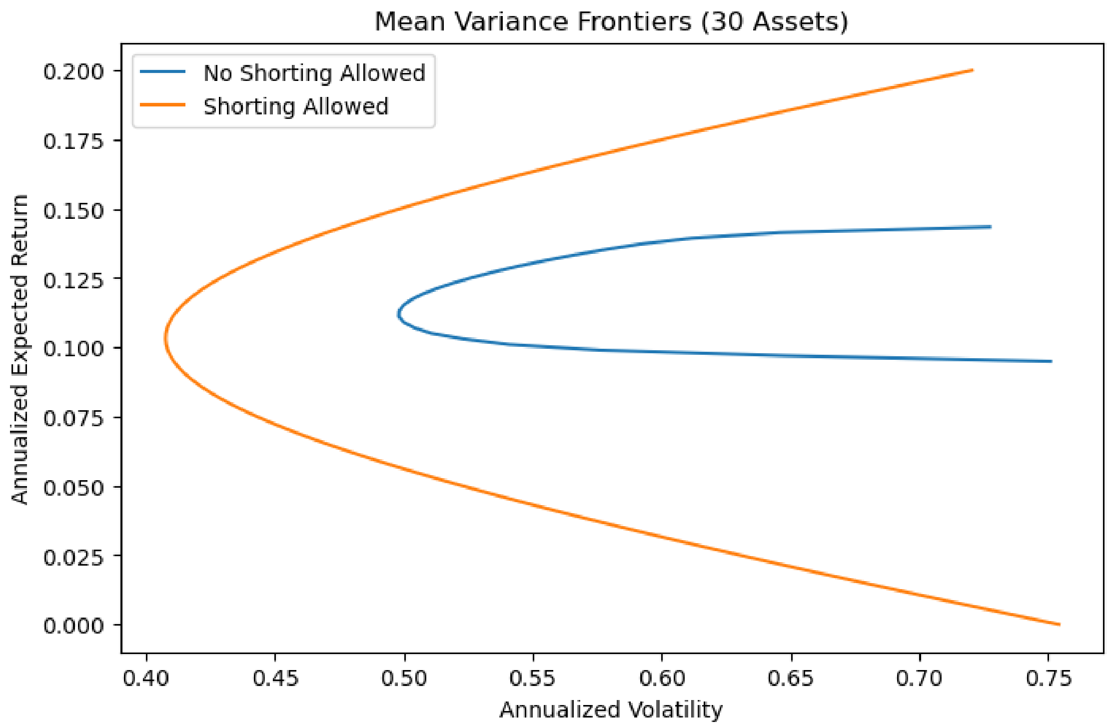
[6]:
```
fig, ax = plt.subplots(figsize=(8, 5))
plt.plot(no_shorting_volatilities, no_shorting_expected_returns, label="No␣
↪Shorting Allowed")
plt.plot(yes_shorting_volatilities, expected_returns, label="Shorting Allowed")
x = np.linspace(0, 1.5, 1000)
plt.plot(x, x * max((expected_returns - rf) / yes_shorting_volatilities) + rf,␣
↪label="Frontier w/ Shorting")
plt.plot(x, x * max((no_shorting_expected_returns - rf) /␣
↪no_shorting_volatilities) + rf, label="Frontier w/o Shorting")
ax.set_title("Mean Variance Frontiers (8 Assets)")
ax.set_xlabel("Annualized Volatility")
ax.set_ylabel("Annualized Expected Return")
ax.legend()
fig.show()
```



1.1.6 2d

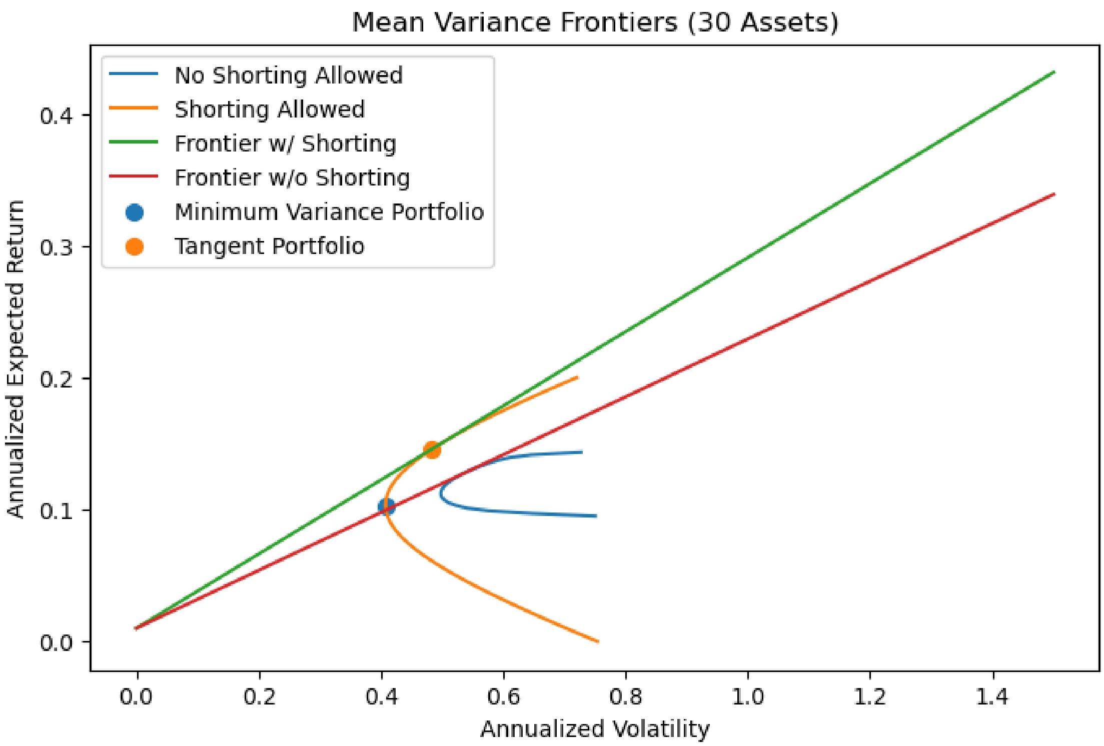
[7]:
```
# Cutting df down to contain our risky assets
# Algorithm: fix desired expected return and use scipy.optimize to minimize␣
↪portfolio variance
expected_returns = np.linspace(0, 0.2, 100)
no_shorting_expected_returns = []
no_shorting_volatilities, yes_shorting_volatilities = [], []

# Store weights here; use this to find the tangent portfolio and minimum␣
↪variance portfolio later on
yes_shorting_weights = []
N = len(df.columns)
mu, Sigma = np.array(df.mean()), np.array(df.cov())

def objective_func(weights, cov_matrix):
    portfolio_volatility = np.sqrt(weights.T @ cov_matrix @ weights)
    return portfolio_volatility


for i, target_return in enumerate(expected_returns):

    # Define constraints
```

```
    sum_to_one_constraint = {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}
    target_return_constraint = {'type': 'eq', 'fun': lambda weights: np.dot(weights, mu) - target_return}
    cons = [target_return_constraint, sum_to_one_constraint]

    # Not allowing shorting case
    w0 = np.ones(N) / N
    res = minimize(objective_func, x0=w0, args=(Sigma,), constraints=cons, bounds=[(0, None)] * N, tol=1e-6)
    if res.success:
        no_shorting_expected_returns.append(target_return)
        no_shorting_volatilities.append(res.fun)

    # Allowing shorting case
    w0 = np.random.random(N)
    w0 /= np.sum(w0)
    res = minimize(objective_func, x0=w0, args=(Sigma,), constraints=cons, tol=1e-6)
    yes_shorting_min_vol = res.fun
    yes_shorting_weights.append(res.x)
    yes_shorting_volatilities.append(yes_shorting_min_vol)

no_shorting_expected_returns = np.array(no_shorting_expected_returns)
no_shorting_volatilities = np.array(no_shorting_volatilities)
yes_shorting_volatilities = np.array(yes_shorting_volatilities)

fig, ax = plt.subplots(figsize=(8, 5))
plt.plot(no_shorting_volatilities, no_shorting_expected_returns, label="No Shorting Allowed")
plt.plot(yes_shorting_volatilities, expected_returns, label="Shorting Allowed")
ax.set_title("Mean Variance Frontiers (30 Assets)")
ax.set_xlabel("Annualized Volatility")
ax.set_ylabel("Annualized Expected Return")
ax.legend()
fig.show()
```



[8]:

```
fig, ax = plt.subplots(figsize=(8, 5))
plt.plot(no_shorting_volatilities, no_shorting_expected_returns, label="No␣
↪Shorting Allowed")
plt.plot(yes_shorting_volatilities, expected_returns, label="Shorting Allowed")
x = np.linspace(0, 1.5, 1000)
plt.plot(x, x * max((expected_returns - rf) / yes_shorting_volatilities) + rf,␣
↪label="Frontier w/ Shorting")
plt.plot(x, x * max((no_shorting_expected_returns - rf) /␣
↪no_shorting_volatilities) + rf, label="Frontier w/o Shorting")
plt.scatter(np.min(yes_shorting_volatilities), expected_returns[np.
↪argmin(yes_shorting_volatilities)], label="Minimum Variance Portfolio", s=50)
plt.scatter(yes_shorting_volatilities[np.argmax(expected_returns /␣
↪yes_shorting_volatilities)], expected_returns[np.argmax(expected_returns /␣
↪yes_shorting_volatilities)], label="Tangent Portfolio", s=50)
ax.set_title("Mean Variance Frontiers (30 Assets)")
ax.set_xlabel("Annualized Volatility")
ax.set_ylabel("Annualized Expected Return")
ax.legend()
fig.show()
```



[9]:
```
yes_shorting_weights[np.argmin(yes_shorting_volatilities)], np.
↪min(yes_shorting_volatilities) ** 2
```

[9]:
```
(array([ 2.99401952e-01,  1.02592554e-03,  7.34056741e-02, -1.26702868e-01,
        -6.21232711e-02,  8.00532058e-02,  1.78711719e-01,  9.61207531e-02,
        -2.72580687e-02,  9.34979293e-03, -9.56966506e-02, -8.27302565e-02,
        -9.59630290e-02, -1.52267431e-01,  5.69890602e-03, -1.19894829e-02,
         1.23233405e-01, -2.96649163e-02,  1.69191864e-01,  1.87181696e-01,
         4.44105738e-01,  5.07201101e-03,  1.04019875e-01,  1.05996190e-01,
         9.70729754e-02, -1.63794262e-03, -1.29125215e-02,  1.20066192e-02,
        -2.92291328e-01, -4.10534842e-04]),
 0.16615018308721213)
```

[10]:
```
yes_shorting_weights[np.argmax(expected_returns / yes_shorting_volatilities)],␣
↪np.max(expected_returns / yes_shorting_volatilities)
```

[10]:
```
(array([ 0.13318001,  0.16263039,  0.25237924, -0.05505554, -0.115541  ,
        -0.0774272 ,  0.15297645,  0.16692424,  0.07941963,  0.03595398,
        -0.29445224, -0.23379319, -0.06283194,  0.01471314,  0.07632006,
         0.13625607,  0.09526605,  0.00969675,  0.22388603,  0.08340817,
         0.26933311,  0.08304199,  0.18683482,  0.12542306, -0.020465  ,
        -0.15901172,  0.0883895 ,  0.06899205, -0.19116015, -0.23528677]),
 0.3011839993167813)
```

1.1.7 2e

I would argue that yes, the tangent portfolio here is statistically noisy. While our method of finding the tangent, optimal portfolio is extremely statistical, I believe that the weights of tangent portfolio themselves are noisy and not robust at all. Consider the covariance matrix of the 30 assets below.

[11]:
```
Sigma, list(filter(lambda x: x < 0, Sigma.flatten()))
```

[11]:
```
(array([[0.32265026, 0.34313587, 0.26366159, 0.4278295 , 0.35605357,
         0.31548488, 0.26707944, 0.29397936, 0.31526355, 0.3546473 ,
         0.36144576, 0.36438435, 0.35497052, 0.3855143 , 0.3583689 ,
         0.36216414, 0.25113264, 0.3653125 , 0.2491698 , 0.26504702,
         0.20418032, 0.23527988, 0.28987007, 0.30295767, 0.34032713,
         0.34380897, 0.32573651, 0.32154586, 0.37026999, 0.33129859],
        [0.34313587, 0.74354876, 0.27395224, 0.59421737, 0.44223381,
         0.39516312, 0.32215072, 0.36776237, 0.39963857, 0.48517259,
         0.48758556, 0.47196961, 0.46693299, 0.48360046, 0.45894895,
         0.48210443, 0.33999325, 0.48084739, 0.31529819, 0.32748924,
         0.2411125 , 0.29527914, 0.37197444, 0.37424455, 0.44552441,
         0.50195803, 0.40041496, 0.42377337, 0.46981374, 0.43720738],
        [0.26366159, 0.27395224, 0.48580438, 0.36272468, 0.28712183,
         0.26610699, 0.2212222 , 0.26022268, 0.26256743, 0.28639604,
         0.30678303, 0.31408893, 0.29736313, 0.31763702, 0.30067033,
         0.30358913, 0.22812007, 0.31873305, 0.2136936 , 0.23365408,
         0.1763462 , 0.2024915 , 0.24574072, 0.26099265, 0.28335701,
         0.29465072, 0.261201  , 0.26417008, 0.30903415, 0.29663635],
        [0.4278295 , 0.59421737, 0.36272468, 1.14488772, 0.67418613,
         0.52726755, 0.48963601, 0.4826263 , 0.58400635, 0.73322823,
         0.69861777, 0.77835725, 0.72177311, 0.74649898, 0.73966099,
         0.7041567 , 0.54082897, 0.74377467, 0.44813901, 0.42868647,
         0.36792138, 0.46631947, 0.62625565, 0.5312307 , 0.67706294,
         0.70686516, 0.58034924, 0.61666816, 0.69527352, 0.66146715],
        [0.35605357, 0.44223381, 0.28712183, 0.67418613, 0.7371574 ,
         0.42542203, 0.42932811, 0.37812887, 0.46007209, 0.60139514,
         0.56983155, 0.6191957 , 0.58095532, 0.58362056, 0.61017599,
         0.56436693, 0.40505178, 0.61287139, 0.36893526, 0.35822367,
         0.29529217, 0.41547176, 0.45636393, 0.45154509, 0.55601288,
         0.54407468, 0.46878209, 0.47095634, 0.54638726, 0.50772856],
        [0.31548488, 0.39516312, 0.26610699, 0.52726755, 0.42542203,
         0.4846294 , 0.30348103, 0.34620688, 0.38481382, 0.42790148,
         0.43937651, 0.45687303, 0.44322219, 0.48286577, 0.46806241,
         0.44114096, 0.30673749, 0.41495858, 0.2874942 , 0.29458415,
         0.22964839, 0.28481893, 0.38184932, 0.37389275, 0.40372941,
         0.41862284, 0.38215919, 0.37258276, 0.43249909, 0.41687779],
        [0.26707944, 0.32215072, 0.2212222 , 0.48963601, 0.42932811,
         0.30348103, 0.53289535, 0.26847196, 0.36136985, 0.49338939,
         0.4349981 , 0.44601515, 0.43059291, 0.41481766, 0.45367527,
         0.43081707, 0.29633404, 0.43335591, 0.25760943, 0.22553718,
```

```
 0.21311877, 0.32269416, 0.3584467 , 0.35059299, 0.40823629,
 0.41714417, 0.38792268, 0.38362424, 0.39272806, 0.39095169],
[0.29397936, 0.36776237, 0.26022268, 0.4826263 , 0.37812887,
 0.34620688, 0.26847196, 0.44561334, 0.35118474, 0.37139086,
 0.38877497, 0.40571812, 0.39369422, 0.43882594, 0.38505371,
 0.39650811, 0.27672438, 0.40528973, 0.27152083, 0.27222931,
 0.22223343, 0.27942564, 0.36768655, 0.33682651, 0.36896316,
 0.38846136, 0.34496759, 0.34496914, 0.4078023 , 0.37033333],
[0.31526355, 0.39963857, 0.26256743, 0.58400635, 0.46007209,
 0.38481382, 0.36136985, 0.35118474, 0.57040324, 0.51546233,
 0.52347753, 0.61121617, 0.56295223, 0.5678513 , 0.57289828,
 0.51602264, 0.42469707, 0.57486314, 0.3811427 , 0.31847771,
 0.27278442, 0.30869765, 0.44708406, 0.45516664, 0.49214029,
 0.4520804 , 0.40612903, 0.39367426, 0.48974145, 0.46216079],
[0.3546473 , 0.48517259, 0.28639604, 0.73322823, 0.60139514,
 0.42790148, 0.49338939, 0.37139086, 0.51546233, 0.84904735,
 0.61671522, 0.67580948, 0.62725612, 0.6072564 , 0.67569763,
 0.59945496, 0.4453033 , 0.6433816 , 0.37349644, 0.34627767,
 0.29442048, 0.36416221, 0.47638584, 0.48135269, 0.59279298,
 0.58007764, 0.49728896, 0.48875958, 0.56561333, 0.54634091],
[0.36144576, 0.48758556, 0.30678303, 0.69861777, 0.56983155,
 0.43937651, 0.4349981 , 0.38877497, 0.52347753, 0.61671522,
 0.68098116, 0.6733084 , 0.63095964, 0.63586084, 0.63270179,
 0.59518292, 0.4619434 , 0.65818026, 0.41397503, 0.37103109,
 0.29698012, 0.38773884, 0.50059609, 0.47643624, 0.57130384,
 0.56969163, 0.47854947, 0.48348242, 0.57642405, 0.53968014],
[0.36438435, 0.47196961, 0.31408893, 0.77835725, 0.6191957 ,
 0.45687303, 0.44601515, 0.40571812, 0.61121617, 0.67580948,
 0.6733084 , 1.02332543, 0.76693633, 0.73805872, 0.74571907,
 0.70827623, 0.63478331, 0.87812359, 0.50574576, 0.38520189,
 0.33970356, 0.399493  , 0.59976418, 0.54326839, 0.67677677,
 0.60309368, 0.49382188, 0.47614062, 0.62107324, 0.58557893],
[0.35497052, 0.46693299, 0.29736313, 0.72177311, 0.58095532,
 0.44322219, 0.43059291, 0.39369422, 0.56295223, 0.62725612,
 0.63095964, 0.76693633, 0.74841275, 0.6796031 , 0.6706738 ,
 0.64881202, 0.51850183, 0.75835914, 0.45820551, 0.3670254 ,
 0.30918414, 0.40260131, 0.55487482, 0.50544786, 0.61364771,
 0.56827975, 0.47373745, 0.47979134, 0.5788424 , 0.54398922],
[0.3855143 , 0.48360046, 0.31763702, 0.74649898, 0.58362056,
 0.48286577, 0.41481766, 0.43882594, 0.5678513 , 0.6072564 ,
 0.63586084, 0.73805872, 0.6796031 , 0.82793024, 0.68168658,
 0.65175346, 0.48037609, 0.6724982 , 0.42580157, 0.40746331,
 0.33287026, 0.39153287, 0.57356415, 0.51642625, 0.61041728,
 0.56502728, 0.50974284, 0.49505284, 0.61530357, 0.56051737],
[0.3583689 , 0.45894895, 0.30067033, 0.73966099, 0.61017599,
 0.46806241, 0.45367527, 0.38505371, 0.57289828, 0.67569763,
 0.63270179, 0.74571907, 0.6706738 , 0.68168658, 0.90772866,
```

```
 0.62969931, 0.46566619, 0.65032102, 0.41291093, 0.37576359,
 0.32329224, 0.36273365, 0.53334535, 0.51872076, 0.59455711,
 0.56311992, 0.51558812, 0.47402509, 0.60075571, 0.55539388],
[0.36216414, 0.48210443, 0.30358913, 0.7041567 , 0.56436693,
 0.44114096, 0.43081707, 0.39650811, 0.51602264, 0.59945496,
 0.59518292, 0.70827623, 0.64881202, 0.65175346, 0.62969931,
 0.82514917, 0.46446016, 0.70056541, 0.42457925, 0.36965647,
 0.2878346 , 0.36973149, 0.49405746, 0.47749357, 0.63122306,
 0.57219747, 0.46122376, 0.49298573, 0.57814292, 0.54299665],
[0.25113264, 0.33999325, 0.22812007, 0.54082897, 0.40505178,
 0.30673749, 0.29633404, 0.27672438, 0.42469707, 0.4453033 ,
 0.4619434 , 0.63478331, 0.51850183, 0.48037609, 0.46566619,
 0.46446016, 0.76384316, 0.64522627, 0.40456068, 0.27185812,
 0.22267992, 0.29491091, 0.37720823, 0.36292026, 0.44285212,
 0.42918679, 0.32018942, 0.35216595, 0.41501044, 0.40608843],
[0.3653125 , 0.48084739, 0.31873305, 0.74377467, 0.61287139,
 0.41495858, 0.43335591, 0.40528973, 0.57486314, 0.6433816 ,
 0.65818026, 0.87812359, 0.75835914, 0.6724982 , 0.65032102,
 0.70056541, 0.64522627, 1.6779511 , 0.56495579, 0.43652669,
 0.30718497, 0.37128885, 0.51380147, 0.50420309, 0.70833529,
 0.63520459, 0.43370805, 0.46446457, 0.61315917, 0.52827095],
[0.2491698 , 0.31529819, 0.2136936 , 0.44813901, 0.36893526,
 0.2874942 , 0.25760943, 0.27152083, 0.3811427 , 0.37349644,
 0.41397503, 0.50574576, 0.45820551, 0.42580157, 0.41291093,
 0.42457925, 0.40456068, 0.56495579, 0.53463193, 0.29137775,
 0.20554179, 0.26114692, 0.32934352, 0.32926432, 0.38475473,
 0.36533166, 0.28985353, 0.29506772, 0.39836659, 0.35749519],
[0.26504702, 0.32748924, 0.23365408, 0.42868647, 0.35822367,
 0.29458415, 0.22553718, 0.27222931, 0.31847771, 0.34627767,
 0.37103109, 0.38520189, 0.3670254 , 0.40746331, 0.37576359,
 0.36965647, 0.27185812, 0.43652669, 0.29137775, 0.43580606,
 0.22806918, 0.22152667, 0.27976182, 0.29160376, 0.35712769,
 0.34842574, 0.29407216, 0.30417617, 0.40858341, 0.33171801],
[0.20418032, 0.2411125 , 0.1763462 , 0.36792138, 0.29529217,
 0.22964839, 0.21311877, 0.22223343, 0.27278442, 0.29442048,
 0.29698012, 0.33970356, 0.30918414, 0.33287026, 0.32329224,
 0.2878346 , 0.22267992, 0.30718497, 0.20554179, 0.22806918,
 0.30358611, 0.24834469, 0.28891992, 0.24427326, 0.29135882,
 0.28068273, 0.26300362, 0.24899132, 0.31798719, 0.27667836],
[0.23527988, 0.29527914, 0.2024915 , 0.46631947, 0.41547176,
 0.28481893, 0.32269416, 0.27942564, 0.30869765, 0.36416221,
 0.38773884, 0.399493  , 0.40260131, 0.39153287, 0.36273365,
 0.36973149, 0.29491091, 0.37128885, 0.26114692, 0.22152667,
 0.24834469, 0.9953588 , 0.44020946, 0.31437234, 0.35319959,
 0.37943059, 0.33017976, 0.36000839, 0.36120712, 0.35832541],
[0.28987007, 0.37197444, 0.24574072, 0.62625565, 0.45636393,
 0.38184932, 0.3584467 , 0.36768655, 0.44708406, 0.47638584,
```

```
        0.50059609, 0.59976418, 0.55487482, 0.57356415, 0.53334535,
        0.49405746, 0.37720823, 0.51380147, 0.32934352, 0.27976182,
        0.28891992, 0.44020946, 0.65642873, 0.41128601, 0.45801409,
        0.46611902, 0.42048675, 0.40473758, 0.47079229, 0.46456294],
       [0.30295767, 0.37424455, 0.26099265, 0.5312307 , 0.45154509,
        0.37389275, 0.35059299, 0.33682651, 0.45516664, 0.48135269,
        0.47643624, 0.54326839, 0.50544786, 0.51642625, 0.51872076,
        0.47749357, 0.36292026, 0.50420309, 0.32926432, 0.29160376,
        0.24427326, 0.31437234, 0.41128601, 0.49442301, 0.46049581,
        0.42128326, 0.38549003, 0.37877953, 0.45943841, 0.43364001],
       [0.34032713, 0.44552441, 0.28335701, 0.67706294, 0.55601288,
        0.40372941, 0.40823629, 0.36896316, 0.49214029, 0.59279298,
        0.57130384, 0.67677677, 0.61364771, 0.61041728, 0.59455711,
        0.63122306, 0.44285212, 0.70833529, 0.38475473, 0.35712769,
        0.29135882, 0.35319959, 0.45801409, 0.46049581, 0.72006846,
        0.54193721, 0.44837751, 0.47658561, 0.56381663, 0.50668134],
       [0.34380897, 0.50195803, 0.29465072, 0.70686516, 0.54407468,
        0.41862284, 0.41714417, 0.38846136, 0.4520804 , 0.58007764,
        0.56969163, 0.60309368, 0.56827975, 0.56502728, 0.56311992,
        0.57219747, 0.42918679, 0.63520459, 0.36533166, 0.34842574,
        0.28068273, 0.37943059, 0.46611902, 0.42128326, 0.54193721,
        0.76951878, 0.46323738, 0.47801459, 0.53749833, 0.52652651],
       [0.32573651, 0.40041496, 0.261201  , 0.58034924, 0.46878209,
        0.38215919, 0.38792268, 0.34496759, 0.40612903, 0.49728896,
        0.47854947, 0.49382188, 0.47373745, 0.50974284, 0.51558812,
        0.46122376, 0.32018942, 0.43370805, 0.28985353, 0.29407216,
        0.26300362, 0.33017976, 0.42048675, 0.38549003, 0.44837751,
        0.46323738, 0.51284463, 0.42853246, 0.47101201, 0.43249798],
       [0.32154586, 0.42377337, 0.26417008, 0.61666816, 0.47095634,
        0.37258276, 0.38362424, 0.34496914, 0.39367426, 0.48875958,
        0.48348242, 0.47614062, 0.47979134, 0.49505284, 0.47402509,
        0.49298573, 0.35216595, 0.46446457, 0.29506772, 0.30417617,
        0.24899132, 0.36000839, 0.40473758, 0.37877953, 0.47658561,
        0.47801459, 0.42853246, 0.60217343, 0.46539155, 0.43813783],
       [0.37026999, 0.46981374, 0.30903415, 0.69527352, 0.54638726,
        0.43249909, 0.39272806, 0.4078023 , 0.48974145, 0.56561333,
        0.57642405, 0.62107324, 0.5788424 , 0.61530357, 0.60075571,
        0.57814292, 0.41501044, 0.61315917, 0.39836659, 0.40858341,
        0.31798719, 0.36120712, 0.47079229, 0.45943841, 0.56381663,
        0.53749833, 0.47101201, 0.46539155, 0.66171266, 0.52645867],
       [0.33129859, 0.43720738, 0.29663635, 0.66146715, 0.50772856,
        0.41687779, 0.39095169, 0.37033333, 0.46216079, 0.54634091,
        0.53968014, 0.58557893, 0.54398922, 0.56051737, 0.55539388,
        0.54299665, 0.40608843, 0.52827095, 0.35749519, 0.33171801,
        0.27667836, 0.35832541, 0.46456294, 0.43364001, 0.50668134,
        0.52652651, 0.43249798, 0.43813783, 0.52645867, 0.65776039]]),
[])
```

There is not a single negative number within our entire covariance matrix. From a economic point of view, this means that all of our assets are postiively correlated with each other. Recall that the variance of our portfolio is given as

$$\mathrm{Var}(R_p) = \sum w_i^2 \sigma_i^2 + 2 \sum_{i<j} w_i w_j \mathrm{Cov}(R_i, R_j)$$

Thus, from an economic point of view, with all positive covariances, we can never decrease the variance of our portfolio, or hedge our positions. We can only choose assets such that the variance increases the least. I argue that this situation is statstically noisy essentially we're heavily weighting assets whose industries have no relation to each other. This is quite random and intuitively kind of noisy. Alternatively, if there was one other asset in our universe with negative covariances with some of the other assets, our tangent portfolio weights would look extremely different our portfolio would be far more truly "diversifed."

1.1.8 3a

[12]:
```
df = pd.read_csv("F-F_Research_Data_Factors_monthly.csv") / 100

def date_helper(date):
    date = str(date)
    return pd.to_datetime(f"{date[:4]}-{date[5:]}")

df["Date"] = df["mon"].apply(date_helper)
mkt_rf = df[["Date", "Mkt-RF", "RF"]]
df = df.drop(columns=["mon"])
df = df.set_index("Date")
ff3 = df.copy()

model = LinearRegression()
X = df["Mkt-RF"].to_numpy().reshape(-1, 1)
y = (df["HML"] - df["RF"]).to_numpy().reshape(-1, 1)
model.fit(X, y)
alpha = model.intercept_[0]
beta = model.coef_[0][0]
average_hml_return = df["HML"].mean()
print(f"Average HML Return={average_hml_return}, Alpha={alpha}, Beta={beta}")
print(f"""Of the Average HML Return, {(average_hml_return - alpha) * 100 /
 ↪average_hml_return}%
      is systematic and {alpha * 100/average_hml_return}% is excess to CAPM""")
```

```
Average HML Return=0.0036754054054054055, Alpha=-8.638113103718714e-05,
Beta=0.1567957935363723
Of the Average HML Return, 102.35024770084266%
      is systematic and -2.350247700842653% is excess to CAPM
```

1.1.9 3b

Beta is positive, so the market portfolio and the returns on the HML strategy are directly correlated. Thus, the strategy does better when the market performs better.

1.1.10 3c

Recall that $\beta = \frac{\text{Cov}(R_i, R_m)}{\text{Var}(R_m)}$, which represents sensitivity to the market index. Thus, $\beta \approx 15.37\%$ of the variability in the HML PF returns can be explained by market wide movements.

1.1.11 3d

Prior to the publication of this strategy, there should be a clear relationship between betas and average excess returns. Almost all of the average HML return should be systematic, or explained by the variation in the market portfolio. Alpha should be close to 0, as this satisfies the CAPM. After the publication of this strategy, more people will try to use it to make money, which will cause multiple things to happen. By running regressions in both time periods, I predict that the post-1993 average HML return will explained less by the market, and there will be al essclear, possibly inverse relationship between betas and expected excess return. The existence of a positive alpha will confirm that the returns of the portfolios are less explained by the market.

[13]:
```
pre_1993 = df[df.index < pd.to_datetime('1993-01')]
model = LinearRegression()
X = pre_1993["Mkt-RF"].to_numpy().reshape(-1, 1)
y = (pre_1993["HML"] - pre_1993["RF"]).to_numpy().reshape(-1, 1)
model.fit(X, y)
alpha = model.intercept_[0]
beta = model.coef_[0][0]
average_hml_return = pre_1993["HML"].mean()
print(f"Average HML Return={average_hml_return}, Alpha={alpha}, Beta={beta}")
print(f"""Of the Average HML Return, {(average_hml_return - alpha) * 100 /␣
 ↪average_hml_return}%
      is systematic and {alpha * 100/average_hml_return}% is excess to CAPM""")
```

```
Average HML Return=0.0043031328320802, Alpha=-0.000141225528949447,
Beta=0.2131097010273286
Of the Average HML Return, 103.28192353014528%
      is systematic and -3.281923530145278% is excess to CAPM
```

[14]:
```
post_1993 = df[df.index >= pd.to_datetime('1993-01')]
X = post_1993["Mkt-RF"].to_numpy().reshape(-1, 1)
y = (post_1993["HML"] - post_1993["RF"]).to_numpy().reshape(-1, 1)
model.fit(X, y)
alpha = model.intercept_[0]
beta = model.coef_[0][0]
average_hml_return = post_1993["HML"].mean()
print(f"Average HML Return={average_hml_return}, Alpha={alpha}, Beta={beta}")
print(f"""Of the Average HML Return, {(average_hml_return - alpha) * 100 /␣
 ↪average_hml_return}%
      is systematic and {alpha * 100/average_hml_return}% is excess to CAPM""")
```

```
Average HML Return=0.002069871794871794, Alpha=0.0007541634013428375,
Beta=-0.10661637411483195
Of the Average HML Return, 63.56472882951915%
```

```
is systematic and 36.43527117048086% is excess to CAPM
```

1.1.12 4a

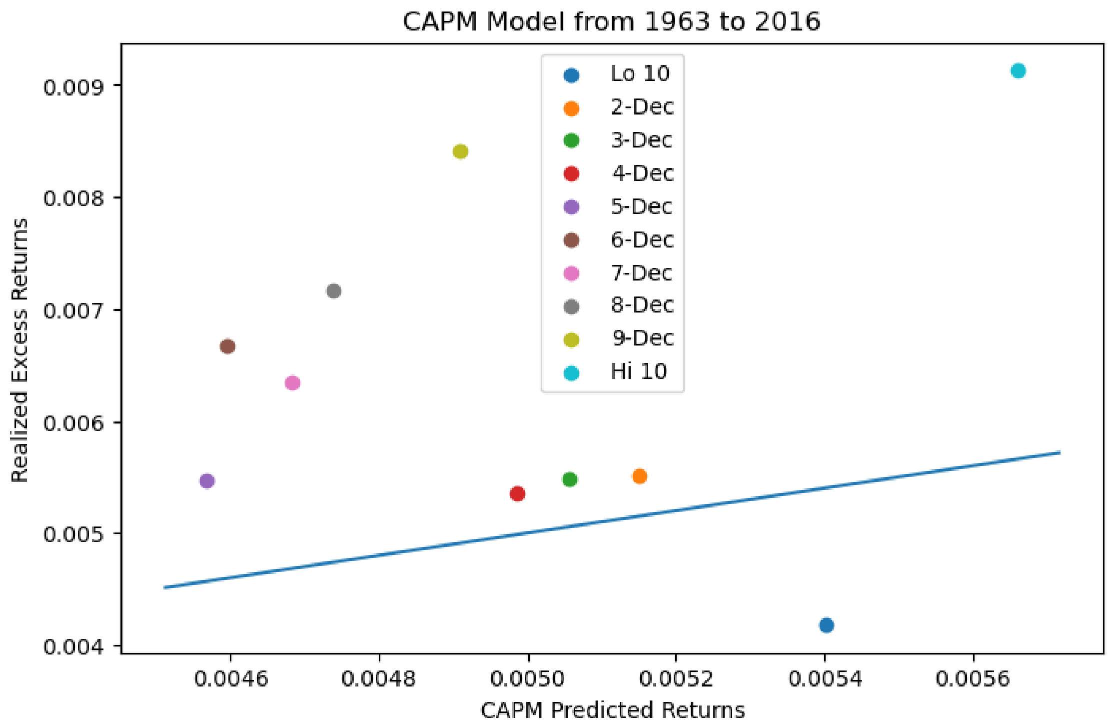
[15]:
```python
df = pd.read_csv("BM_PF_10_mon_vw_PA2.csv") / 100

def date_helper(date):
    date = str(date)
    return pd.to_datetime(f"{date[:4]}-{date[4:]}")

df["Date"] = df["mon"].apply(date_helper)
df = df.drop(columns=["mon"])
df = df.set_index("Date")

# Running CAPM regressions on the ten B/M portfolios on excess market returns
df_a = df[(df.index >= pd.to_datetime('1963-01')) & (df.index < pd.
 ↪to_datetime('2016-01'))]
mkt_a = mkt_rf[(mkt_rf["Date"] >= pd.to_datetime('1963-01')) & (mkt_rf["Date"]␣
 ↪< pd.to_datetime('2016-01'))]

# Sample Averages
print(f"Sample Averages: \n{df_a.mean()}")

# Run the Linear Regressions on all 10 portfolios
model = LinearRegression()
bm_portfolios = df.columns
alphas_capm, betas_capm = [], []
for bm in bm_portfolios:
    # Define X and Y for regression
    X = mkt_a["Mkt-RF"].to_numpy().reshape(-1, 1)
    y = (df_a[bm].values - mkt_a["RF"].values).reshape(-1, 1)
    model.fit(X, y)
    alphas_capm.append(model.intercept_[0])
    betas_capm.append(model.coef_[0][0])

for bm, alpha, beta in zip(bm_portfolios, alphas_capm, betas_capm):
    print(f"B/M: {bm}, Alpha = {alpha}, Beta = {beta}")
```

```
Sample Averages:
Lo 10     0.008159
2-Dec     0.009498
3-Dec     0.009460
4-Dec     0.009330
5-Dec     0.009449
6-Dec     0.010657
7-Dec     0.010330
8-Dec     0.011144
9-Dec     0.012392
```

```
Hi 10    0.013110
dtype: float64
B/M: Lo 10, Alpha = -0.0012233581523273399, Beta = 1.058945874647164
B/M: 2-Dec, Alpha = 0.00036917680440186514, Beta = 1.0093396882205552
B/M: 3-Dec, Alpha = 0.00042502117825842303, Beta = 0.9909654597262043
B/M: 4-Dec, Alpha = 0.0003648539760164049, Beta = 0.9772708103475208
B/M: 5-Dec, Alpha = 0.0009016597076053181, Beta = 0.895285836763551
B/M: 6-Dec, Alpha = 0.0020826333667589396, Beta = 0.9005563640329516
B/M: 7-Dec, Alpha = 0.0016677143615263365, Beta = 0.9178739031218796
B/M: 8-Dec, Alpha = 0.0024258446506329285, Beta = 0.9288241392459646
B/M: 9-Dec, Alpha = 0.0035020370781546515, Beta = 0.9624004247961663
B/M: Hi 10, Alpha = 0.0034697569008832717, Beta = 1.1096081523264008
```

1.1.13 4b

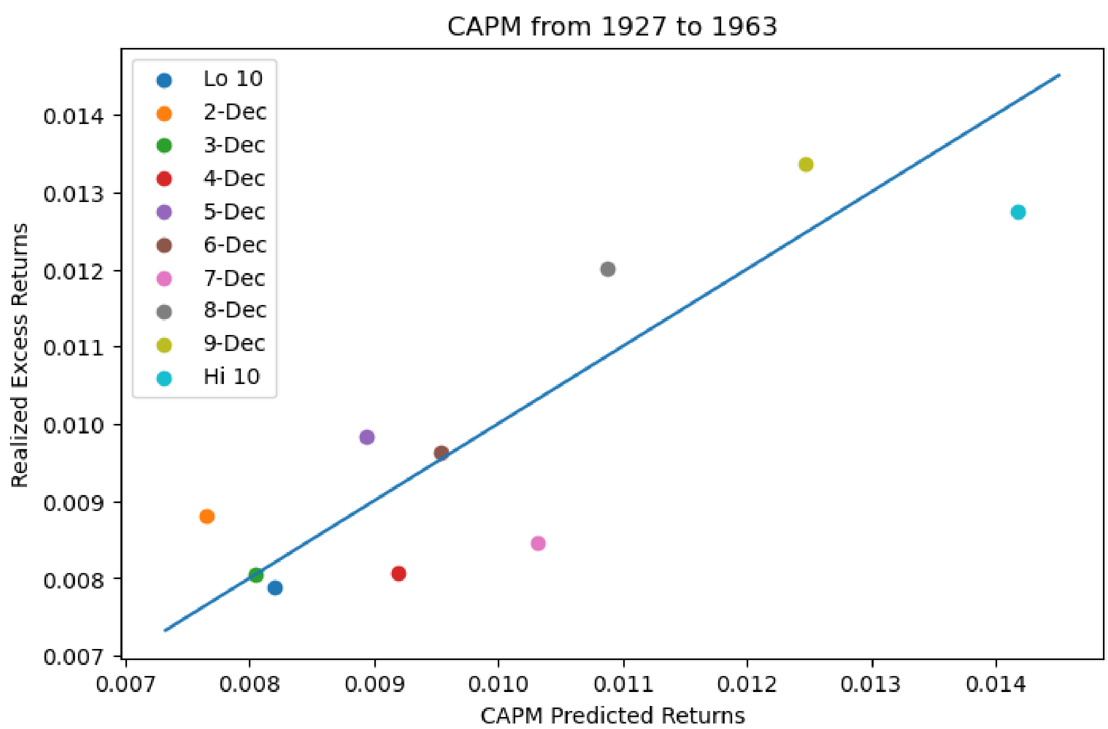
[16]:
```
fig, ax = plt.subplots(figsize=(8, 5))
for bm, beta in zip(bm_portfolios, betas_capm):
    plt.scatter(x=np.mean(beta * mkt_a['Mkt-RF']), y=(df_a[bm].values -␣
 ↪mkt_a["RF"].values).mean(), label=bm)

ax.legend()
plt.xlabel("CAPM Predicted Returns")
x = np.linspace(*ax.get_xlim())
ax.plot(x, x)
plt.ylabel("Realized Excess Returns")
plt.title("CAPM Model from 1963 to 2016")
fig.show()
```



1.1.14 4c

[23]:
```
# Get the time periods for part c too
df_c = df[(df.index >= pd.to_datetime('1927-01')) & (df.index < pd.
 ↪to_datetime('1963-01'))]
mkt_c = mkt_rf[(mkt_rf["Date"] >= pd.to_datetime('1927-01')) & (mkt_rf["Date"]␣
 ↪< pd.to_datetime('1963-01'))]

# Run the Linear Regressions on all 10 portfolios
model = LinearRegression()
bm_portfolios = df.columns
alphas_capm_modern, betas_capm_modern = [], []
for bm in bm_portfolios:
    # Define X and Y for regression
    X = mkt_c["Mkt-RF"].to_numpy().reshape(-1, 1)
    y = (df_c[bm].values - mkt_c["RF"].values).reshape(-1, 1)
    model.fit(X, y)
    alphas_capm_modern.append(model.intercept_[0])
    betas_capm_modern.append(model.coef_[0][0])

fig, ax = plt.subplots(figsize=(8, 5))
for bm, beta in zip(bm_portfolios, betas_capm_modern):
```

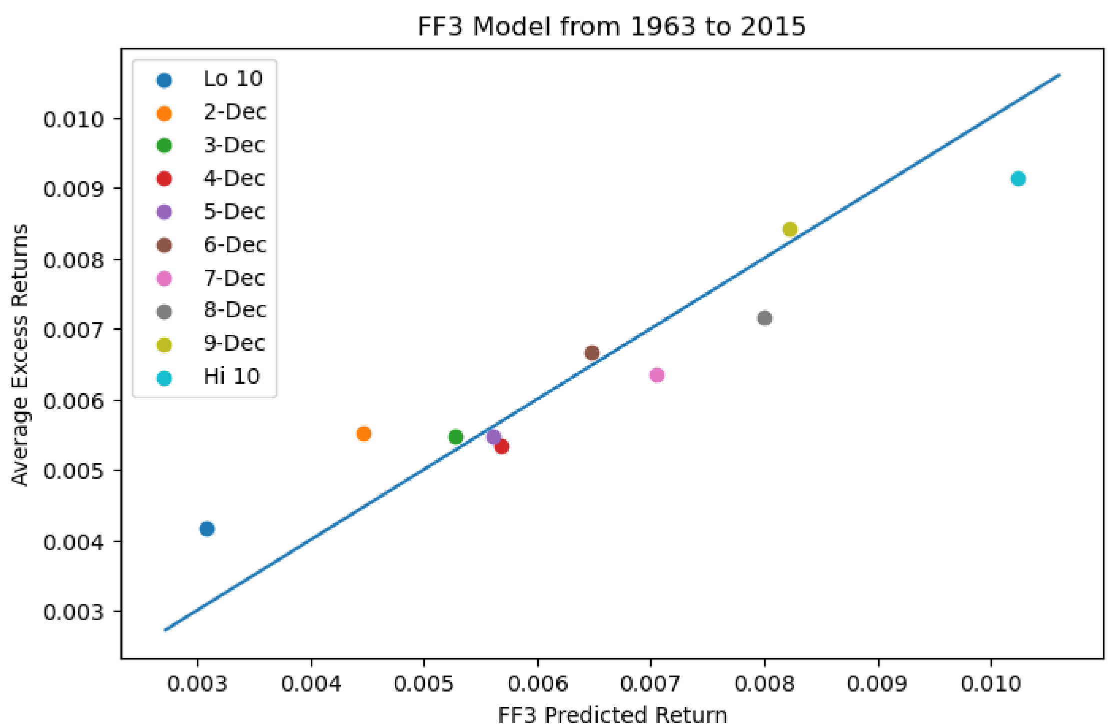
```
    plt.scatter(x=np.mean(beta * mkt_c["Mkt-RF"]), y=(df_c[bm].values -␣
↪mkt_c["RF"].values).mean(), label=bm)

ax.legend()
x = np.linspace(*ax.get_xlim())
ax.plot(x, x)
plt.xlabel("CAPM Predicted Returns")
plt.ylabel("Realized Excess Returns")
plt.title("CAPM from 1927 to 1963")
fig.show()
```



1.1.15 4d

[26]:
```
df_d = df[(df.index >= pd.to_datetime('1963-01')) & (df.index < pd.
↪to_datetime('2016-01'))]
ff3_d = ff3[(ff3.index >= pd.to_datetime('1963-01')) & (ff3.index < pd.
↪to_datetime('2016-01'))]

alphas_ff3, betas_ff3 = [], []

model = LinearRegression()
for bm in bm_portfolios:
    X = (ff3_d[["Mkt-RF", "SMB", "HML"]]).to_numpy()
```

```
    y = (df_d[bm] - ff3_d["RF"]).to_numpy().reshape(-1, 1)
    model.fit(X, y)
    alphas_ff3.append(model.intercept_[0])
    betas_ff3.append(model.coef_[0])

fig, ax = plt.subplots(figsize=(8, 5))
for bm, (beta_rf, beta_smb, beta_hml), alpha in zip(bm_portfolios, betas_ff3,␣
 ↪alphas_ff3):
    ff3_predicted =  beta_rf * ff3_d["Mkt-RF"] + beta_smb * ff3_d["SMB"] +␣
 ↪beta_hml * ff3_d["HML"]
    ff3_predicted = np.mean(ff3_predicted)
    plt.scatter(x=ff3_predicted, y=(df_d[bm].values - mkt_a["RF"].values).
 ↪mean(), label=bm)
ax.legend()

x = np.linspace(*ax.get_xlim())
ax.plot(x, x)
plt.xlabel("FF3 Predicted Return")
plt.ylabel("Average Excess Returns")
plt.title("FF3 Model from 1963 to 2015")
fig.show()
```



1.1.16 4e

Comparing the alphas and factor loadings between CAPM and FF3

[25]:
```
for bm, alpha_capm, alpha_ff3, beta_capm, beta_ff3 in zip(bm_portfolios,␣
↪alphas_capm, alphas_ff3, betas_capm, betas_ff3):
    print(f"B/M: {bm}, CAPM alpha={alpha_capm}, FF3 alpha={alpha_ff3}     |    ␣
↪CAPM beta={beta_capm}, FF3 betas={beta_ff3}")
```

```
B/M: Lo 10, CAPM alpha=-0.0012233581523273399, FF3 alpha=0.0010957173814620138
|     CAPM beta=1.058945874647164, FF3 betas=[ 1.00263484 -0.12521275
-0.49123721]
B/M: 2-Dec, CAPM alpha=0.00036917680440186514, FF3 alpha=0.0010543667453606059
|     CAPM beta=1.0093396882205552, FF3 betas=[ 0.99582013 -0.0499139
-0.14215417]
B/M: 3-Dec, CAPM alpha=0.00042502117825842303, FF3 alpha=0.00020131412277852277
|     CAPM beta=0.9909654597262043, FF3 betas=[ 1.00364713 -0.01795906
0.05432921]
B/M: 4-Dec, CAPM alpha=0.0003648539760164049, FF3 alpha=-0.0003368252617736685
|     CAPM beta=0.9772708103475208, FF3 betas=[ 1.00655135 -0.0128392
0.16035685]
B/M: 5-Dec, CAPM alpha=0.0009016597076053181, FF3 alpha=-0.0001487902304552726
|     CAPM beta=0.895285836763551, FF3 betas=[ 0.95064022 -0.06695131
0.25109433]
B/M: 6-Dec, CAPM alpha=0.0020826333667589396, FF3 alpha=0.0002056124308094271
|     CAPM beta=0.9005563640329516, FF3 betas=[ 0.97735671 -0.02802147
0.42749956]
B/M: 7-Dec, CAPM alpha=0.0016677143615263365, FF3 alpha=-0.0007063577750120063
|     CAPM beta=0.9178739031218796, FF3 betas=[1.00532537 0.00469121 0.53142919]
B/M: 8-Dec, CAPM alpha=0.0024258446506329285, FF3 alpha=-0.0008341119458560968
|     CAPM beta=0.9288241392459646, FF3 betas=[1.00984585 0.16828807 0.69232448]
B/M: 9-Dec, CAPM alpha=0.0035020370781546515, FF3 alpha=0.0001893493978787318
|     CAPM beta=0.9624004247961663, FF3 betas=[1.03656435 0.20485404 0.69570081]
B/M: Hi 10, CAPM alpha=0.0034697569008832717, FF3 alpha=-0.001109809305716995
|     CAPM beta=1.1096081523264008, FF3 betas=[1.17016381 0.45709471 0.92156636]
```