HW 1

Team Members:

Brandt Lawson

Nick Ding

Gurleen Kaur

Michael Bushnell

Joyce Liu

Load the Data

- I am using a jupyter notebook, with data stored in the same folder as this file. Since the data is stored in the same folder, I can use a relative path for the data, steping into this files folder with a '.', and then provinging the data file's name.

In [1068]:
```
import pandas as pd
import numpy as np
from IPython.display import HTML

path_to_data_file = './assetclass_data_monthly_2009.xlsx'
raw_data = pd.read_excel(path_to_data_file)
```

- Calculate excess returns to be used in calculations

In [1069]:
```
raw_data = raw_data.set_index('Dates')
columns_risky_assets = raw_data.columns[0:-1]
returns = raw_data.loc[:,columns_risky_assets]
rf = raw_data['Cash']

excess_returns = returns.subtract(rf, axis=0)

excess_returns.tail(3)
```

Out[1069]:

Dates	Domestic Equity	Foreign Equity	Emerging Markets	Private Equity	Absolute Return	High Yield	Commodities	Real Estate	Domestic Bonds	Foreign Bonds	Inflation-Indexed
2019-07-31	0.013501	-0.021091	-0.028182	-0.004970	-0.002933	-0.000054	-0.003736	0.022205	-0.001231	-0.014542	0.001540
2019-08-30	-0.019007	-0.021504	-0.040085	-0.013188	-0.005225	0.004637	-0.058492	0.032080	0.037227	0.016247	0.021039
2019-09-30	0.018047	0.030209	0.015476	0.021125	0.002858	0.002970	0.016060	0.017476	-0.013294	-0.011568	-0.012491

Q1. Summary Statistics

(a) Calculate and display the mean and volatility of each asset's excess return. (Recall we use volatility to refer to standard deviation.)

In [1070]:
```
mu = excess_returns.mean()
vol = excess_returns.std()
sharpe = mu / vol
summary = pd.DataFrame({'Mean':mu, 'Vol':vol, 'Sharpe': sharpe})
summary.sort_values(by=['Sharpe'],ascending=False)
```

Out[1070]:

	Mean	Vol	Sharpe
Domestic Equity	0.013028	0.037414	0.348220
High Yield	0.007353	0.023927	0.307293
Real Estate	0.014564	0.050511	0.288341
Private Equity	0.013641	0.057616	0.236753
Inflation-Indexed	0.002988	0.013942	0.214291
Domestic Bonds	0.003093	0.016780	0.184348
Foreign Equity	0.008126	0.045516	0.178535
Absolute Return	0.001936	0.012769	0.151593
Emerging Markets	0.008033	0.058230	0.137954
Foreign Bonds	0.002109	0.022195	0.095042
Commodities	-0.001676	0.055118	-0.030401

(b) Which assets have the best and worst Sharpe ratios?

- In the chart above, the asset classes are ranked by descending Sharpe ratios. Domestic Equity, HY Bonds, and Real Estate have the highest Sharpe ratios, while emerging markets, foreign bonds, and commodities have the lowest Sharpe ratios.

Q2. MV Tangency Portfolio

Here we take advantage of the formula for the tangency: $\boldsymbol{w}^{t} \sim \Sigma^{-1}\tilde{\mu}$

Then simply adjust the vector to add up to 1, by dividing it by the sum of its unscaled elements.

(a) Compute and display the weights of the tangent portfolios w^{t}

In [1071]:
```
def compute_tangency(retsx):

    mu = retsx.mean()
    Sigma = retsx.cov()
    Sigma_inv = np.linalg.inv(Sigma)

    weights = Sigma_inv @ mu

    weights = weights / weights.sum()

    wts_tan = pd.DataFrame(weights, index=mu.index)
    wts_tan.columns = ['Weight of asset class']

    retsx_tan = retsx @ wts_tan
    sharpe_tan = retsx_tan.mean()/retsx_tan.std()

    return wts_tan, sharpe_tan
```

In [1072]:
```
Sigma = excess_returns.cov()
Sigma_inv = np.linalg.inv(Sigma)

weights, Q2a_sharpe = compute_tangency(excess_returns)
display(weights)
```

	Weight of asset class
Domestic Equity	1.100132
Foreign Equity	-0.045800
Emerging Markets	-0.144565
Private Equity	-0.166304
Absolute Return	-1.166062
High Yield	0.791084
Commodities	-0.117513
Real Estate	-0.215180
Domestic Bonds	0.799114
Foreign Bonds	-0.022817
Inflation-Indexed	0.187910

(b) Compute the mean, volatility, and Sharpe ratio for the tangency.

In [1073]:
```
def mean_vol_Sharpe_tangency(excess_returns, weights):
    excess_returns_tan = excess_returns @ weights
    mu_tan = excess_returns_tan.mean()
    vol_tan = excess_returns_tan.std()
    sharpe_tan = mu_tan / vol_tan
    table3 = pd.DataFrame([mu_tan, vol_tan, sharpe_tan]).transpose()
    table3.columns = ['Mean','Vol', 'Sharpe']
    table3 = HTML(table3.to_html(index=False))
    return mu_tan, vol_tan, sharpe_tan, table3
Q2_mean, Q2_vol, Q2_sharpe, Q2_table = mean_vol_Sharpe_tangency(excess_returns, weights)
Q2_table
```

Out[1073]:

Mean	Vol	Sharpe
0.014139	0.020714	0.682568

As expected, the Tangency portfolio has a much higher Sharpe Ratio than the individual assets.

If an investor wants a higher or lower mean return, he/she can simply mix this tangency portfolio with the risk-free rate.

Q3. Allocation

(a) Compute and display the weights of MV portfolios with target returns of mu^p = .0067

In [1074]:
```
def mv_target_return_calc(target_return, mu_rf ,mu_tan):
    #target_weights = [(target_return-mu_tan)/(mu_rf-mu_tan)*100, (1-(target_return-mu_tan)/(mu_rf-mu_tan))*100]
    target_weights = (target_return*100/mu_tan, (1-target_return/mu_tan)*100)
    target_weights = ['{:.2f}%'.format(element) for element in target_weights]

    allocation_table1 = pd.DataFrame(target_weights).transpose()
    allocation_table1.columns = ['Risk Free Allocation','Tangent Portfolio Allocation']
    allocation_table1 = HTML(allocation_table1.to_html(index=False))
    return allocation_table1

mu_rf = raw_data['Cash'].mean()
mv_target_return_calc(.0067, mu_rf, float(excess_returns_tan.mean()))
```

Out[1074]:

Risk Free Allocation	Tangent Portfolio Allocation
47.39%	52.61%

(b) What is the mean, volatility, and Sharpe ratio for wp?

- We know the mean (target from (a)), and we know that the sharpe ratio is the same for all linear combinations of the tangent portfolio and risk free rate, and hence the sharpe ratio for wp at target return mu=.0067 = sharpe of tangent portfolio. From these 2 known values, we can calculate the portfolio volatility.

In [1075]:
```
def mean_vol_Sharpe_wp(sharpe_tan, target_return):
    summary_stats_wp = .0067, float(.0067/sharpe_tan), float(sharpe_tan)
    table2 = pd.DataFrame(summary_stats_wp).transpose()
    table2.columns = ['wp mean','wp volatility', 'wp Sharpe']
    table2 = HTML(table2.to_html(index=False))
    display(table2)
    return summary_stats_wp, table2

Q3_mean_tan, Q3_vol_tan, Q3_sharpe_tan, Q3_summary_table = mean_vol_Sharpe_tangency(excess_returns, weights)
Q3_summary_stats_wp, Q3_table = mean_vol_Sharpe_wp(Q3_sharpe_tan, .0067)
```

wp mean	wp volatility	wp Sharpe
0.0067	0.009816	0.682568

(c) Discuss the allocation

- As we can see below, Domestic Equity, Bonds, and HY Bonds are very long, Absolute Return assets are very short, and all other assets are only slightly short, or long (TIPS).

In [1076]:
```
weights.sort_values(by=['Weight of asset class'],ascending=False)
```

Out[1076]:

	Weight of asset class
Domestic Equity	1.100132
Domestic Bonds	0.799114
High Yield	0.791084
Inflation-Indexed	0.187910
Foreign Bonds	-0.022817
Foreign Equity	-0.045800
Commodities	-0.117513
Emerging Markets	-0.144565
Private Equity	-0.166304
Real Estate	-0.215180
Absolute Return	-1.166062

(d) Does this line up with which assets have the strongest Sharpe ratios?

- Lets join the asset class Sharpe ratios and their weights in the tangent portfolio into 1 table.

- While Domestic equity lines of with this logic, i.e. domestic equity has both highest Sharpe and highest allocation, many other asset classes do not follow this pattern. Real estate for example is the second most short asset class, however its Sharpe ratio is the 3rd highest of all the asset classes. Absolute Return is very short, however its Sharpe ratio is lower middle of the pack.

In [1077]:
```
result = pd.merge(weights, summary['Sharpe'], left_index=True, right_index=True)
result.sort_values(by=['Weight of asset class'],ascending=False)
```

Out[1077]:

	Weight of asset class	Sharpe
Domestic Equity	1.100132	0.348220
Domestic Bonds	0.799114	0.184348
High Yield	0.791084	0.307293
Inflation-Indexed	0.187910	0.214291
Foreign Bonds	-0.022817	0.095042
Foreign Equity	-0.045800	0.178535
Commodities	-0.117513	-0.030401
Emerging Markets	-0.144565	0.137954
Private Equity	-0.166304	0.236753
Real Estate	-0.215180	0.288341
Absolute Return	-1.166062	0.151593

Q4. Long-Short positions

(a) Consider an allocation between only domestic and foreign equities. (Drop all other return columns and recompute wp for p = .0067)

In [1078]:
```
excess_returns_dropped = excess_returns.loc[:,['Domestic Equity', 'Foreign Equity']]

#compute_tangency(excess_returns_dropped)

Q4a_weights, Q4a_sharpe = compute_tangency(excess_returns_dropped)
port_mean =  Q4a_weights * (.0067 / (excess_returns_dropped.mean() @ Q4a_weights))
print(port_mean)


#display(Q4_weights)
#excess_returns_dropped

#excess_returns_dropped
```

```
                 Weight of asset class
Domestic Equity               0.769695
Foreign Equity               -0.409531
```

(b) What is causing the extreme long-short position?

- The big difference in historical Sharpe ratios is making the tangent portfolio want to go very short Foreign Equity and very long Domestic Equity.

(c) Make an adjustment to foreign equities of +0.001, (+0.012 annualized.) Recompute wp for p = :0067 for these two assets. How does the allocation among the two assets change?

- Foreign Equities are less short, however they are still very short and Domestic Equities are still very long.

In [1079]:
```
excess_returns_adjusted = excess_returns_dropped
excess_returns_adjusted.loc[:,['Foreign Equity']] = excess_returns_adjusted.loc[:,['Foreign Equity']]+.001

Q4a_weights, Q4a_sharpe = compute_tangency(excess_returns_adjusted)

port_mean =  Q4a_weights * (.0067 / (excess_returns_adjusted.mean() @ Q4a_weights))
print(port_mean)

#Q4c_weights, Q4c_sharpe = compute_tangency(excess_returns_adjusted)
#display(Q4c_weights)
```

```
                 Weight of asset class
Domestic Equity               0.780259
Foreign Equity               -0.379738
```

(d) What does this say about the statistical precision of the MV solutions?

- It means that a small change or error in the estimation of returns of a secuirity will have a large impact on the allocation weights, and hence the viability of th mean-varience model

Q5. Robustness

(a) Recalculate the full allocation, again with the unadjusted μ^(foreign equities) and again for μ^p = 0.0067. This time, make one change: in building w^tan, do not use Σ as given in the formulas in the lecture. Rather, use a diagonalized Σ^D, which zeroes out all non-diagonal elements of the full covariance matrix, Σ. How does the allocation look now?

In [1080]:
```
import sympy as sympy

def compute_tangency_with_diag(retsx):

    mu = retsx.mean()
    Sigma = retsx.cov()
    #diagonal
    Sigma = np.diag(np.diag(np.array(Sigma))) #keep the main diaganal values and sett all other element to 0
    Sigma_inv = np.linalg.inv(Sigma)
    weights = Sigma_inv @ mu
    weights = weights / weights.sum()
    wts_tan = pd.DataFrame(weights, index=mu.index)
    wts_tan.columns=['Weight of asset class']
    retsx_tan = retsx @ wts_tan
    sharpe_tan = retsx_tan.mean()/retsx_tan.std()

    return wts_tan, sharpe_tan

Q5_wts_tan, Q5_sharpe_tan = compute_tangency_with_diag(excess_returns)
display(Q5_wts_tan)
```

	Weight of asset class
Domestic Equity	0.116022
Foreign Equity	0.048898
Emerging Markets	0.029533
Private Equity	0.051224
Absolute Return	0.148000
High Yield	0.160098
Commodities	-0.006876
Real Estate	0.071161
Domestic Bonds	0.136951
Foreign Bonds	0.053381
Inflation-Indexed	0.191609

(b) What does this suggest about the sensitivity of the solution to estimated means and estimated covariances?

- This suggests that the tangent portfolio is extremely sensitive to estimated means and estimated covariances. In other words, a very small error in the estimation of returns and covariance can drastically change the tangent portfolio, which is troubling when searching for a robust way to model a system.

(c) HMC deals with this sensitivity by using explicit constraints on the allocation vector. Conceptually, what are the pros/cons of doing that versus modifying the formula with Σ^D?

Pros

- You can control for specific boundaries of interest i.e. no shorting
- You can ensure that you stay within your firm's investment policy allocations limits, or mandate

Cons

- Worse objective function. Putting bounds on a solution may degrade what you are optimizing over; Adding constraints garentees that the objective funtion (here the Sharpe ratio) will be less than or equal to an unconstrained solution.

6. Out-of-Sample Performance

Let's divide the sample to both compute a portfolio and then check its performance out of sample.

(a) Using only data through the end of 2016, compute w^p for μ^p = .0067, allocating to all 11 assets.

In [1081]:
```
excess_returns_09_16 = excess_returns.loc['2009':'2016',:]
wts_tan_2016, sharpe_tan_2016 = compute_tangency(excess_returns_09_16)

excess_returns_09_16_tan = excess_returns_09_16 @ wts_tan_2016
mu_tan_16 = excess_returns_09_16_tan.mean()

display(mv_target_return_calc(.0067, mu_rf ,float(mu_tan_16)))
display(wts_tan_2016)
```

Risk Free Allocation	Tangent Portfolio Allocation
52.65%	47.35%

	Weight of asset class
Domestic Equity	0.892357
Foreign Equity	-0.078261
Emerging Markets	-0.149629
Private Equity	-0.118068
Absolute Return	-0.647109
High Yield	0.604045
Commodities	-0.070434
Real Estate	-0.160209
Domestic Bonds	0.564514
Foreign Bonds	-0.118201
Inflation-Indexed	0.280995

(b) Calculate the portfolio's Sharpe ratio within that sample, through the end of 2016.

In [1082]:
```
print('Sharpe ratio of 2009-2016 mv portfolio = ' + '{:.4f}'.format(float(sharpe_tan_2016)))
```

```
Sharpe ratio of 2009-2016 mv portfolio = 0.7521
```

(c) Calculate the portfolio's Sharpe ratio based on performance in 2017-2019.

In [1083]:
```
excess_returns_17_19 = excess_returns.loc['2017':,:]
Q6_mean, Q6_vol, Q6_sharpe, Q6_table = mean_vol_Sharpe_tangency(excess_returns_17_19, wts_tan_2016)

print('Sharpe ratio of 2017-2019 mv portfolio = ' + '{:.4f}'.format(Q6_sharpe[0]))
```

```
Sharpe ratio of 2017-2019 mv portfolio = 0.4551
```

(d) How does this out-of-sample Sharpe compare to the 2009-2016 performance of a portfolio optimized to μ^p using 2009-2016 data?

Sharpe much lower (.221582 vs .7521). Really bad OOS preformance

In [1084]:
```
Q6_table = pd.DataFrame([float(sharpe_tan_2016), Q6_sharpe[0]], index = ['in sample', 'out-of-sample'])
Q6_table.columns = (['MV Sharpe'])
Q6_table
```

Out[1084]:

	MV Sharpe
in sample	0.752065
out-of-sample	0.455068

7. Robust Out-of-Sample Performance

Recalculate wp on 2009-2016 data using the diagonalized covariance matrix, Σ^D. What is the performance of this portfolio in 2017-2019? Does it do better out of sample than the portfolio constructed on 2009-2016 data using the full covariance matrix?

In [1085]:
```
excess_returns_09_16 = excess_returns.loc['2009':'2016',:]
wts_tan_diag_2016, sharpe_tan_diag_2016 = compute_tangency_with_diag(excess_returns_09_16)

excess_returns_17_19 = excess_returns.loc['2017':,:]
Q7_mean, Q7_vol, Q7_sharpe, Q7_table = mean_vol_Sharpe_tangency(excess_returns_17_19, wts_tan_diag_2016)

Q6_table['Diag MV Sharp'] = [float(sharpe_tan_diag_2016), Q7_sharpe[0]]

Q6_table
```

Out[1085]:

	MV Sharpe	Diag MV Sharp
in sample	0.752065	0.357775
out-of-sample	0.455068	0.282957

- Using the Diagonal covarence matrix when calculating the tangent portfolio greatly increases the robustness of the model

- As shown in the table above, the 'MV Sharpe' has a very high Sharpe ratio in sample, but when it is applied out-of-sample, the sharpe ratio is very low. The opposite is true with 'diag MV Sharpe'; While the in sample Sharpe is lower than its MV Sharpe counterpart, the out-of-sample Sharpe is only suddely lower than the in sample.

- This implied there is huge over-fitting when we use the entire Covarence Matrix in the calculations.

Source: https://numpy.org/doc/stable/reference/generated/numpy.diag.html

In [ ]: