Creating the Optimal Investment Portfolio

For years, the world has been excited about the opportunities of artificial intelligence in the field of finance. Largely, it's because financial datasets are inherently large and complex making predicting investment trends near impossible without the assistance of programs. However, an easier problem to solve is the allocation of assets in an investment portfolios. Companies like Wealthfront have been touting investment robots that can help allocate investments using key investment principles like CAPM, Mean Variance Optimization, and Black Litterman.

Following a careful evaluation of wealthfront's investment white paper and their methodology, I've come to the conclusion that creating a Black Litterman portfolio optimizer is well within my means.

Every investor is looking to answer several questions:

1. What assets to buy?
2. When to buy said assets?
3. How much of each asset to buy?
4. When to sell the assets?

A portfolio optimizer is mainly used to solve the third question; by optimizing the weights of each asset in the portfolio to maximize the sharpe ratio. However, what's more about Black Litterman is that it enables investors to combine their views regarding the performance of various assets with the market equilibrium in a manner that results in intuitive, diversified portfolios.

Sharpe ratio = (Mean portfolio return − Risk-free rate)/Standard deviation of portfolio return.

$$= \frac{\bar{r}_p - r_f}{\sigma_p}$$

Where:

$\bar{r}_p$ = Expected porfolio return

r_f = Risk free rate

σ_p = Portfolio standard deviation

Background

In order to build the optimizer, a more in depth understanding regarding Black Litterman model is required. Many of the material in this section is taken from the step by step guide to using Black Litterman.

Black Litterman was created to address three main problems with the traditional mean variance optimization proposed as part of Harry Markowitz's Modern Portfolio Theory: 1. Highly concentrated portfolios, 2. Input sensitivity, 3. Estimation error maximization.

Mean variance optimized portfolios often yield extreme long short positions, and when constrained for only long positions, concentrates holdings in few assets. As seen in the case study, when given 6 assets to allocate for a portfolio, the resulting mean variance optimized portfolio concentrated its allocation on VTI and VIG.

	VTI	VEA	VWO	VIG	XLE
Expected Returns	0.167492	0.067939	0.035935	0.147877	0.106458

Covariance Matrix

	VTI	VEA	VWO	VIG	XLE
VTI	0.026690	0.030565	0.031271	0.022685	0.031603
VEA	0.030565	0.042698	0.040840	0.026362	0.037466
VWO	0.031271	0.040840	0.050083	0.026864	0.039206
VIG	0.022685	0.026362	0.026864	0.020282	0.027133
XLE	0.031603	0.037466	0.039206	0.027133	0.048352

	VTI	VEA	VWO	VIG	XLE
Weights_MVO	0.472223	5.898060e-17	0.000000	0.527777	0.000000

Input sensitivity refers to the fact that when using mean variance optimization, the resulting portfolio weights see massive changes when the expected returns of the assets change slightly. Black Litterman overcomes this by allowing investors to incorporate their own views to adjust the expected return. Finally, according to the guide and its cited report has solved the problem of error maximization by spreading out the errors across the expected return vector. (I don't fully understand this last error, so I'm going to simply assume it is correct.)

Process Part 1- Mean Variance Optimization and Implied Equilibrium Return
A lot of the code takes inspiration from
http://www.quantandfinancial.com/2013/08/black-litterman.html
However, I worked out most of the functions on my own, while making some adjustments later on.

Following some research into financial data sources and development communities. I identified Quantopian as my development tool and data source. Quantopian has accurate minute level

data of over 8000 equities, a large data marketplace for additional data along with a web based IDE. This allows for easier access to data, with less data manipulation involved.

Following some early reading through Quantopian's documentation, I realized that its zipline allowed for the direct retrieval of pricing data for most US stocks and ETFs. However, Quantopian's data comes as pricing data and not returns data. So I wrote a function that manipulates the data into percent returns called load_returns_data. It also concatenates the various returns data for a list of symbols into a single dataframe.

```
def load_returns_data(symbols, start, end):
    data={}
    for symbol in symbols:
        data[symbol]=get_pricing(symbol, start_date=start, end_date=end, symbol_reference_date=Non
e, frequency='daily', fields='price', handle_missing='raise', start_offset=0)
    returns={}
    for symbol in symbols:
        returns[symbol] = data[symbol].pct_change()[1:]
    returns_df=pd.DataFrame.from_dict(returns, orient='columns', dtype=None)
    returns_df=returns_df[symbols]
    return returns_df
```

Next, the data provided is fairly cleaned on Quantopian, and there will be a minimal amount of data wrangling required as such. However, checking for NaN values will be required, as a covariance matrix will require that all assets have the same number of returns data during the start and end dates.

Then I needed to first write a function that would be able to optimize the portfolios based on mean variance, that function is solve_weights. Solve_weights optimizes the portfolio's Sharpe ratio. The function also required a list of expected returns and its covariance matrix, which led to me writing the function, assetmeanvar.

```
def solve_weights(R, C, Rf, low=0., high=1): #set low and high for upper and lower bounds for weig
ht
    n= len(R)
    W= np.ones([n])/n
    b_=[(low, high) for i in range(n)]
    c_ = ({'type':'eq', 'fun': lambda W: sum(W)-1. })
    optimized = scipy.optimize.minimize(w_fitness, W, (R, C, Rf), method='SLSQP', constraints=c_,
bounds=b_)
    if not optimized.success: #copied
        raise BaseException(optimized.message)
    return optimized.x
```

```
def assetmeanvar(returns):
    covars=np.cov(np.matrix(returns).T)*250
    (rows, cols)= np.shape(returns)
    expectedreturns= np.array([])
    for i in range(cols):
        expectedreturns=np.append(expectedreturns,(1+np.mean(returns.iloc[:,i]))**250-1)
    return covars, expectedreturns
```

The first line Weights_MVO shows the resulting weights calculated by the function solve_weights. This is the result when mean variance optimization is utilized. The high concentration of asset weights can be observed by seeing the weights of VTI and VIG.

```
Rf=0.015  #setting the risk free rate at 1.5%
W1_mvo=solve_weights(R1, C1, Rf)
display(pd.DataFrame([W1_mvo, W1], columns=returns_df1.columns, index= ['Weights_MVO', 'Weights_ma
rket_cap']))
```

	VTI	VEA	VWO	VIG	XLE
Weights_MVO	0.472223	5.898060e-17	0.000000	0.527777	0.000000
Weights_market_cap	0.729591	1.156200e-01	0.099084	0.037259	0.018446

Black Litterman creates an equilibrium return by reverse optimization.

$$\Pi = \lambda \Sigma w_{mkt} \qquad (1)$$

where

Π is the Implied Excess Equilibrium Return Vector ($N \times 1$ column vector);
λ is the risk aversion coefficient;
Σ is the covariance matrix of excess returns ($N \times N$ matrix); and,
w_{mkt} is the market capitalization weight ($N \times 1$ column vector) of the assets.[4]

The equilibrium is achieved by setting the initial “optimal” weight as the market cap weights, as seen as w-mkt. This allows every asset weight to have a portfolio allocation. This requires a weight by market cap; even though Quantopian offers market cap information for equities, ETF’s net asset values are not available, thus each ETF’s NAV needs to be hardcoded.

```
Market_cap1={'VTI': 656960000000.000, 'VEA':104110000000.000, 'VWO': 89220000000.000, 'VIG':335500
00000.000, 'XLE':16610000000.000}
Market_cap2={'VTI': 656960000000.000, 'VEA':104110000000.000, 'VWO': 89220000000.000, 'VIG':335500
00000.000, 'VNQ':34630000000.000, 'LQD':38460000000.000, 'EMB':12470000000.000}
```

The next component that needs to be calculated is the risk aversion coefficient. “[it] is the expected risk-return tradeoff. It is the rate at which an investor will forego expected return for less variance. In the reverse optimization process, the risk aversion coefficient acts as a scaling factor for the reverse optimization estimate of excess returns; the weighted reverse optimized excess returns equal the specified market risk premium. More excess return per unit of risk (a larger lambda) increases the estimated excess returns”

$$\lambda = \frac{E(r) - r_f}{\sigma^2}$$

where

$E(r)$ is the expected market (or benchmark) total return;

r_f is the risk-free rate; and,

$\sigma^2 = w_{mkt}^T \Sigma w_{mkt}$ is the variance of the market (or benchmark) excess returns.

This formula is thus used to write the function lam.

```
def lam(m, v, Rf):
    """
    Calculates the risk aversion coefficient
    Requires: expected (market/ benchmark) return,
    variance of the (market/benchmark) returns,
    and risk free rate
    """
    lam= (m-Rf)/v
    return lam
```

The benchmark return in this case is used by calculating the expected return of the MVO portfolio and its variance. Which required the creation of function mean_var, which calculates the portfolio expected return and variance.

```
def mean_var(W, R, C):
    """
    Calculates portfolio expected return and variance
    Requires: List of weights, list of asset expected returns, and asset covariance matrix.
    """
    mean= compute_mean(W,R)
    var= compute_var(W,C)
    return mean, var
```

Once we have lambda and market weights, we can calculate for the equilibrium return.

```
def eqret(C, W, lam):
    """
    Calculates the implied equilibrium excess return by finding the dot products of lam, covariance matrix, a
    Requires: Covariance matrix, market weights, and lambda
    """
    eqret= np.dot(np.dot(lam,C),W)
    return eqret
```

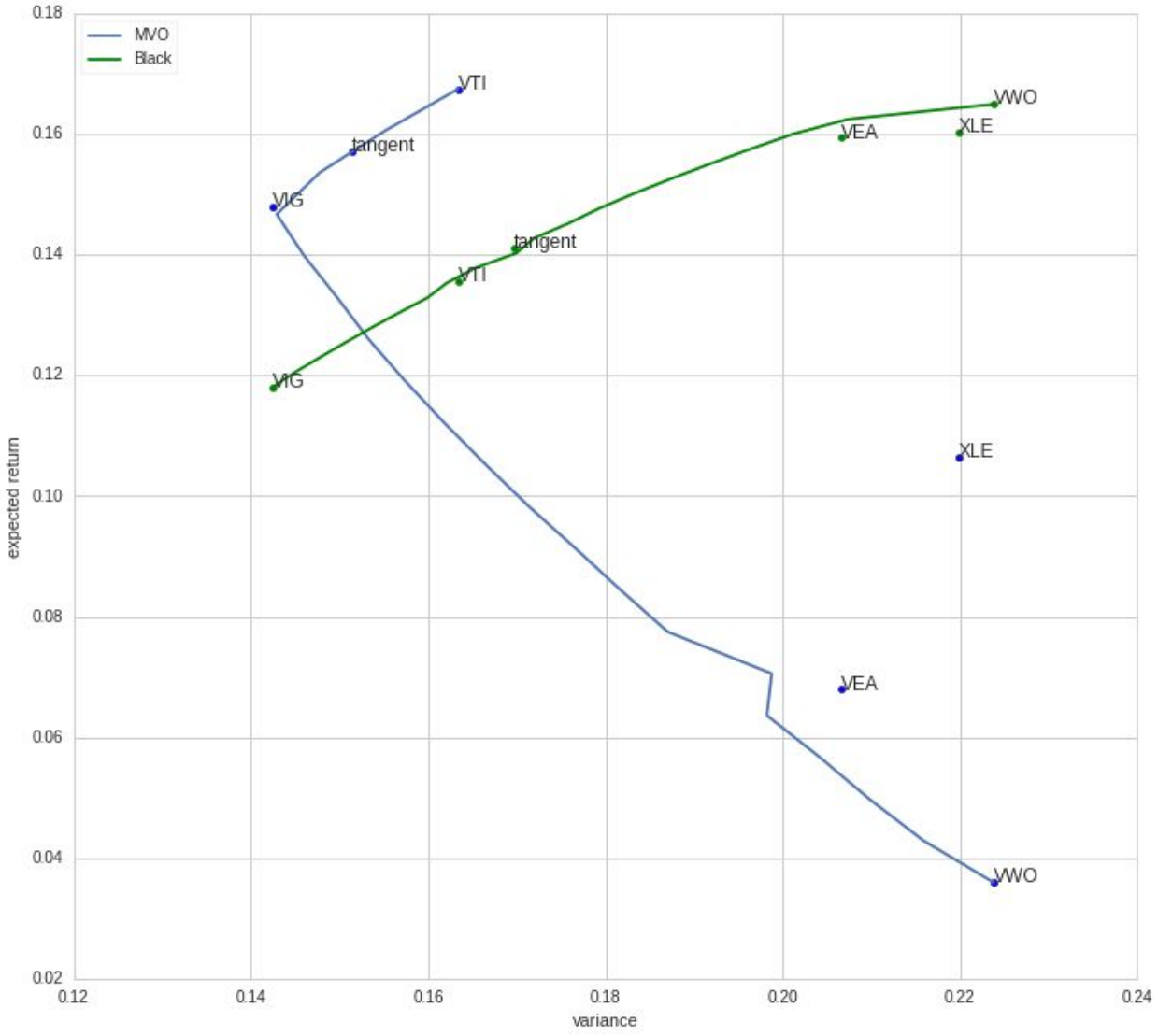
Which yields an implied equilibrium return list. Where we can see differences between the expected returns and the equilibrium returns.

	VTI	VEA	VWO	VIG	XLE
Expected Returns	0.167492	0.067939	0.035935	0.147877	0.106458
Equilibrium Returns	0.120637	0.144399	0.149952	0.103038	0.145366

By adding the risk free rates to the equilibrium returns we can calculate for the new equilibrium weights.

```
W1_eq=solve_weights(eqret1+Rf, C1, Rf)
```

	VTI	VEA	VWO	VIG	XLE
Weights_MVO	0.472223	5.898060e-17	0.000000	0.527777	0.000000
Weights_market_cap	0.729591	1.156200e-01	0.099084	0.037259	0.018446
Equilibrium Weights	0.730696	1.155927e-01	0.098778	0.036141	0.018792



Through the graph, we can see how the expected(historical) returns form a efficient frontier (blue). Each point on the frontier represents a portfolio composition that minimizes variance at the specific expected return. The tangent portfolio on the blue curve is the MVO portfolio. We can also see how the curve is smoothed out through using reverse optimization with market weights in the first step of Black Litterman, and how the expected returns of each asset has shifted. It's important to note each asset's variance remains constant. The tangent point on the green curve represents the Equilibrium portfolio.

	VTI	VEA	VWO	VIG	XLE
Weights_MVO	0.472223	5.898060e-17	0.000000	0.527777	0.000000
Weights_market_cap	0.729591	1.156200e-01	0.099084	0.037259	0.018446
Equilibrium Weights	0.730696	1.155927e-01	0.098778	0.036141	0.018792

	expected_return	variance
MVO	0.157140	0.151355
Equilibrium	0.141108	0.169665

As seen above, the equilibrium weights is very close to the weights by market cap. This is due to the fact that the weights equilibrium returns are derived from the market weights, and as such the equilibrium weights are only adjusted with the risk free return, thus the two are nearly identical.

Process Part 2- Incorporating Views

$$E[R] = \left[(\tau\Sigma)^{-1} + P'\Omega^{-1}P\right]^{-1}\left[(\tau\Sigma)^{-1}\Pi + P'\Omega^{-1}Q\right] \quad (3)$$

where

$E[R]$ is the new (posterior) Combined Return Vector (N x 1 column vector);
τ is a scalar;
Σ is the covariance matrix of excess returns (N x N matrix);
P is a matrix that identifies the assets involved in the views (K x N matrix or 1 x N row vector in the special case of 1 view);

Ω is a diagonal covariance matrix of error terms from the expressed views representing the uncertainty in each view (K x K matrix);
Π is the Implied Equilibrium Return Vector (N x 1 column vector); and,
Q is the View Vector (K x 1 column vector).

Tau is static and can be approximated with 0.025.

The greatest advantage offered by Black Letterman's model is the ability to incorporate views about the varying assets against one another. The original MVO model requires you to incorporate views for every single asset, meaning, you have to input a specific expected return for each asset. This isn't very practical as traders and portfolio managers often have views of the assets in relation to each other. For example, I could believe that Google will outperform Apple, but I wouldn't know if Google can outperform the market and by how much.

The a views function must be able to transform a list of views into Q, a views vector (K x 1), and P, a matrix that links the assets involved in the views.
Example list of views:

```
views1 = [('VEA', '>','VWO', 0.03),
          ('XLE','VTI', '>', 'VIG', 0.02),
          ('VWO','<', 'XLE', 0.02)]
```

The views vector is straightforward, but the links matrix has certain complexities. If there are only two assets being compared, the links matrix is fairly straightforward, it would only have two non-zero elements similar to row 1. However, when you have more than 2 assets being related, as seen in the second view in views1, they needed to be weighted some how. The resulting second row is created using the market weights. The weighting is broken down by weighting the positive and negative sides separately, the weight for the link in relation to VTI will be calculated by VTI's market cap divided by the sum of VTI and XLE's market cap, vice versa. This allows the link matrix to better reflect the effect of the views. The resulting function is views_and_links.

```
 the views matrix for portfolio 1
 [ 0.03   0.02   0.02]

the links matrix for portfolio 1
[[ 0.          1.         -1.          0.          0.        ]
 [ 0.97534035  0.          0.         -1.          0.02465965]
 [ 0.          0.         -1.          0.          1.        ]]
```

Then we create a diagonal covariance matrix of error terms from the expressed views representing the uncertainty in each view (K x K matrix).

General Case:

$$\Omega = \begin{bmatrix} \omega_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \omega_k \end{bmatrix}$$

Determining the individual variances of the error terms (ω) that constitute the diagonal elements of Ω is one of the most complicated aspects of the model. I simply used the calculation:

$$\omega = \tau \cdot \mathbf{P} \cdot \mathbf{C} \cdot \mathbf{P}^T$$

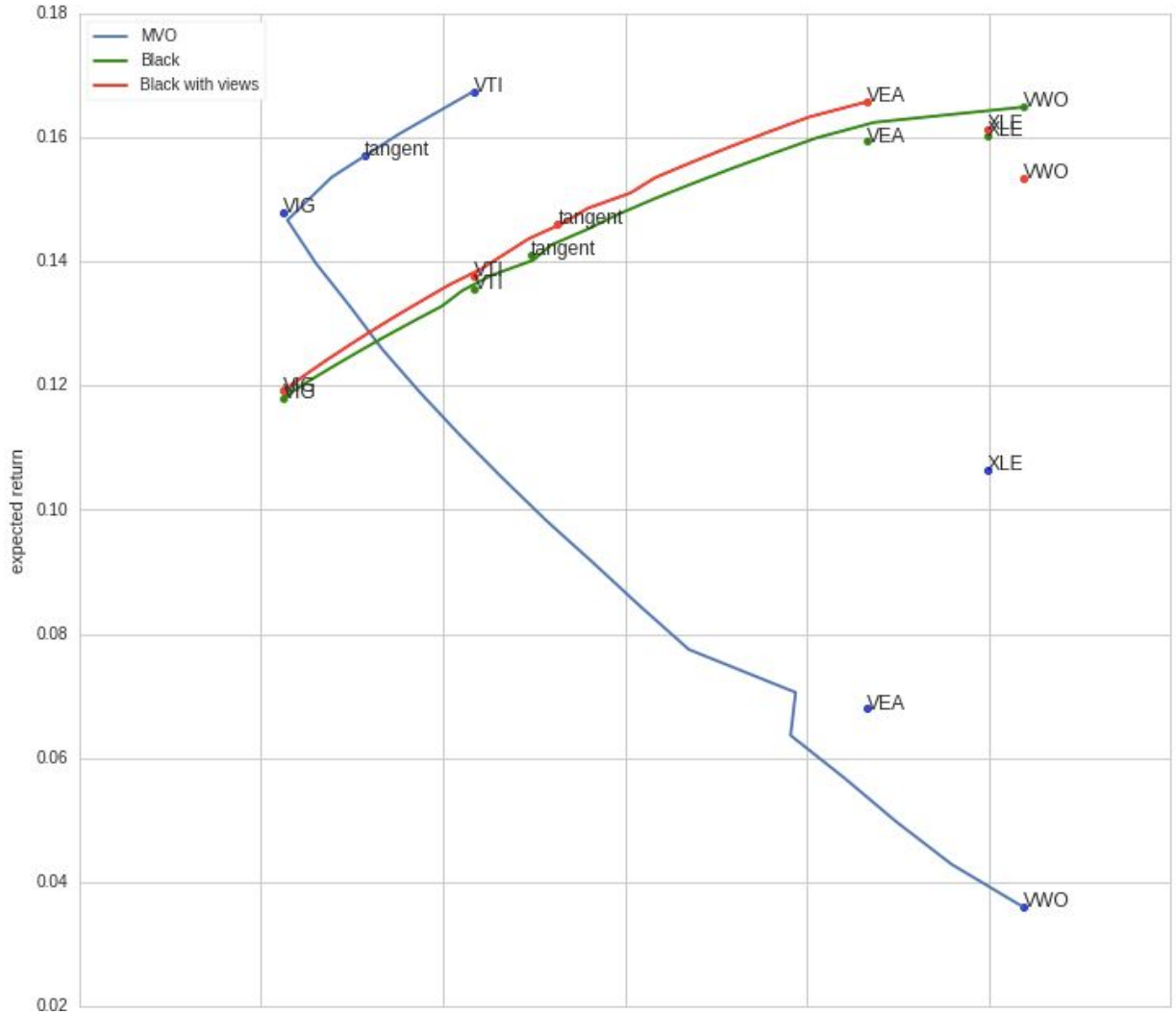
Which created the function uncertainty_m.

Finally, we simply need to combine all the components to have the combined return vector. Which is the new expected returns accounting for the user imported views. Leads to the creation of function eqexcessret. Yielding the following uncertainty matrix and equilibrium returns adjusted for views.

```
uncertainty matrix for portfolio 1
[[  2.77537635e-04  -5.74638849e-06   1.87586896e-04]
 [ -5.74638849e-06   4.08134043e-05   7.01832220e-06]
 [  1.87586896e-04   7.01832220e-06   5.00577817e-04]]
equilibrium returns adjusted for views
```

	VTI	VEA	VWO	VIG	XLE
Equilibrium Returns	0.122698	0.150822	0.138599	0.104176	0.146305

Once we have the new equilibrium returns, we will solve for the optimal portfolio weights, returns, and variance.



As seen from the graph, the views shifted the black litterman curve from the green to the red. Showing that our views were actually able to change the expected returns, and yielding a new tangent portfolio on the red curve.