CQF Final Project – Portfolio Construction

Gaurav Sharma

Table of Contents

Executive Summary

I have chosen "Portfolio Construction with Robust Covariance" as the topic for CQF final project. The aim is to replicate and expand the liquid US ETF called '*The RPAR Risk Parity ETF'* which invests in five diverse asset classes- Global equities, Commodities, Gold, TIPS and Treasuries. The portfolio choice in this project is closest to Option B. The RPAR Risk parity ETF can potentially earn equity-like returns over the long run with less volatility and reduced risk of material loss because of its superior balance across various asset classes which perform well in different economic environments. We have added three more asset classes in the mix, namely- US Investment Grade Bonds, US High Yield Bonds, and Agency MBS. Addition of these asset classes provide more diversification benefit considering their negative correlation with some of the existing five asset classes and these three assets have higher sharpe ratios than existing five asset classes. (Web-link for RPAR ETF intro slides- https://www.rparetf.com/rpar/IntrotoRPARRiskParityETF)

After calculating excess returns, vols, naïve covariance, and correlations among 8 asset classes, we used Marcenko-Pastur Denoising to get the robust covariance as denoising technique is superior than Ledoit-Wolf nonlinear shrinkage in terms of preservation of signal carried by top eigenvectors. We chose to keep top 3 eigenvalues as is and averaging the bottom five eigenvalues to arrive at denoised covariance and correlations.

We used historical excess returns as expected returns and naïve/robust covariances in constrained mean variance optimizer with RPAR ETF (benchmark) implied lambda (risk aversion coefficient) to get the optimal portfolio allocation for both naïve vs robust covariances. The constraints used are as follows- a) sum of weights should be 1, and 2) individual weights should be in a range of (0.05, 0.3) in order to avoid corner solutions/extremely concentrated portfolios and getting final optimal allocations with representation from broader asset classes which perform well under different economic environments.

As a next step, we implemented Black-Litterman model, using index weights suggested by RPAR ETF, albeit allocating 10% each from Treasuries and TIPS to newly added three asset classes (US-IG, US-HY, and Agency MBS), thus rotating some fixed income allocations in the portfolio. The two views used in BL model are- 1) Treasuries will underperform Equities by 5% and 2) Gold will have absolute excess returns of 10%. Incorporating these two views, new BL allocations are calculated using constrained optimization (sum of weights is 1 and individual weights in (0.05,0.3) range) for mean-variance, min variance, and max sharpe optimizers, using robust covariances and market implied lambda (risk aversion coefficient). Similarly, BL allocations are calculated using constrained optimization (sum of weights is 1 and individual weights in (0.05,0.3) range) for mean variance optimizer for three different risk-aversion coefficients (lambdas = 1.4, 7, 12.1) to study the relationship between risk aversion coefficient and portfolio stats like returns, risk, and Sharpe ratios.

Finally, for conclusion, we selected the optimal allocation out of all the above scenarios which delivers the best results (in terms of balanced allocation, good diversification, and better Sharpe ratio) thus enhancing the current index weights suggested by RPAR ETF, incorporating our asset class views and weight constraints.

Raw Data

We downloaded the daily price data for eight asset classes using Bloomberg in a spreadsheet named 'Price_data.xls', which is further used in python code. In order to run the python code properly and using the price data the path of file location of 'Price_data.xls' saved at local PC should be pasted in the code (screenshot below).

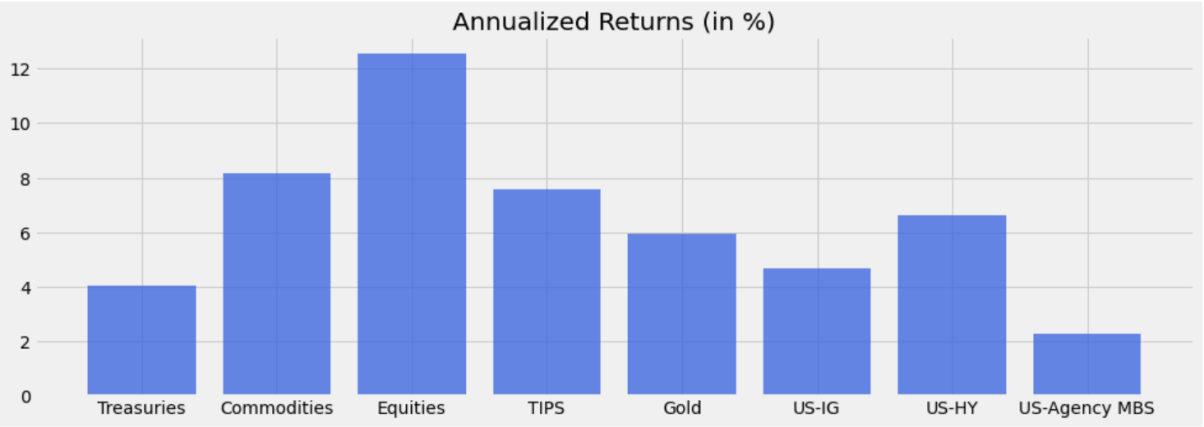
```
#Reading raw data from spreadsheet

file =(r'C:\Users\Gunjan\Desktop\Price data.xls')
```

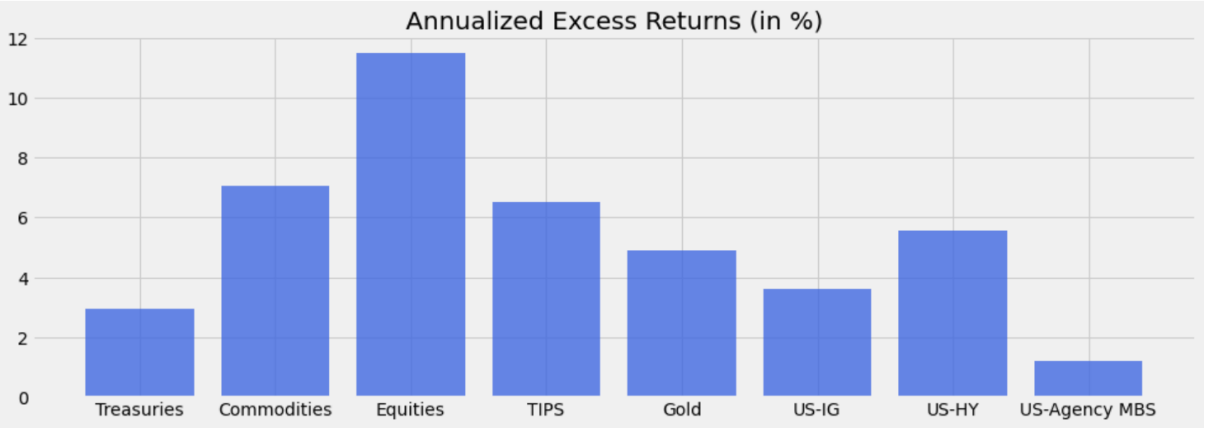
The five asset classes and related price indices which are used in RPAR ETF are shown in the screenshot below. The additional three asset classes daily price data (US-Investment Grade Bonds, US-High yield Bonds, and Agency MBS) is downloaded from Bloomberg and pasted in the 'Price_data.xls' spreadsheet. We also downloaded the daily price data for 3M treasury bills (proxy for risk free rate) to calculate historical excess returns for all asset classes. Further, we have used the past five-year daily data for optimal portfolio allocation calculations.

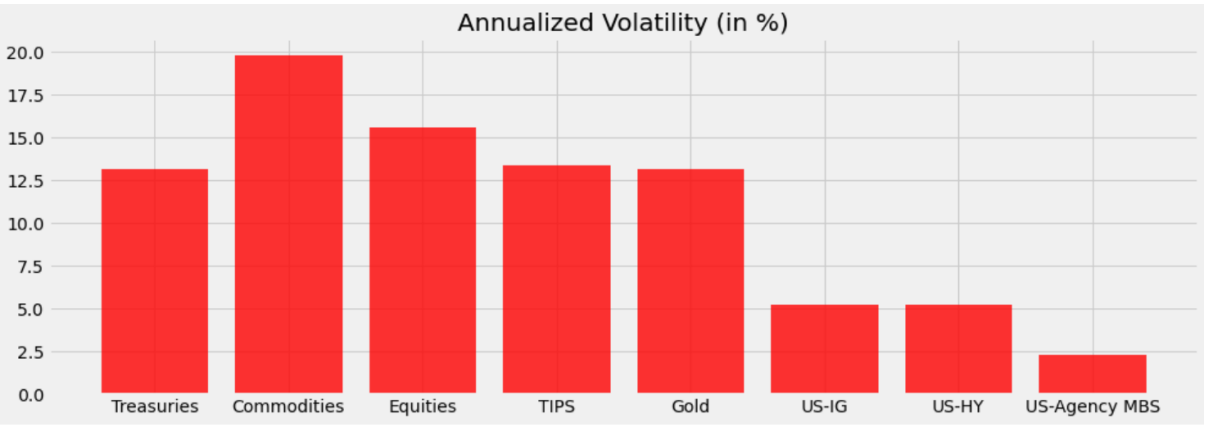
As of 6/30/21	Index
Global Equities	MSCI World Index
Treasuries	Bloomberg Barclays Long Treasury Index
TIPS	Merrill Lynch 15+ Yr. US Inflation-Linked Bond Index
Commodity Producers	S&P Global Natural Resources Index
Physical Gold	Spot Price of Gold (% change)

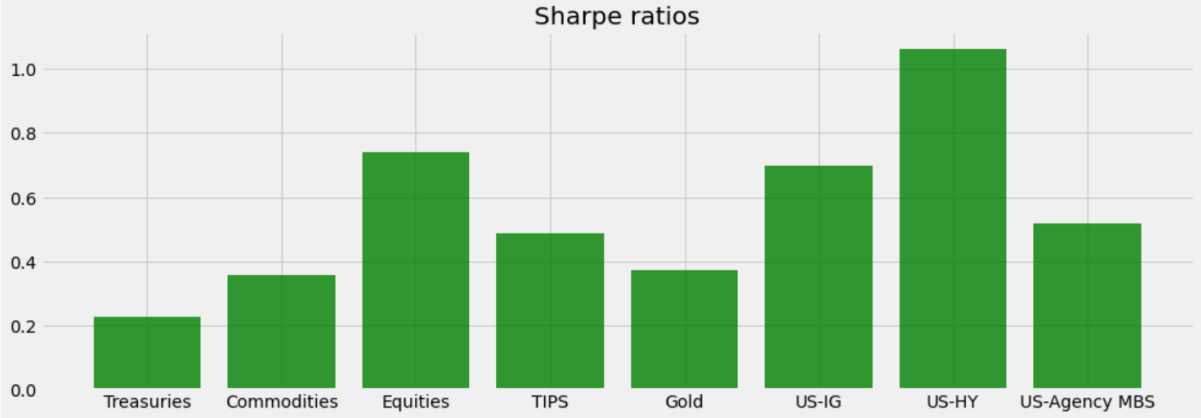
Charts (returns, excess returns, vols, and sharpe ratio)

The calculated annualized returns, annualized excess returns, annualized vols, and sharpe ratios for 8 asset classes in the portfolio are shown in the charts below-









Python code for importing libraries and using raw data to calculate historical returns, vols, and sharpe ratios

```
# importing libraries

import numpy as np
import array as arr
import pandas as pd
import scipy as sci
import scipy.optimize as sco
import seaborn as sn
import datetime
import sklearn
from numpy import *
from numpy.linalg import multi_dot
from scipy.linalg import eigh, cholesky
from scipy.stats import norm
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

```
#Reading raw data from spreadsheet

file =(r'C:\Users\Gunjan\Desktop\Price data.xls')
rawData = pd.read_excel(file)
rawData

df = pd.DataFrame(rawData)
df[["Date1", "Date2", "Date3", "Date4", "Date5", "Date6", "Date7", "Date8", "Date9"]] = df[["Date1", "Date2", "Date3", "Date4",

df1 = df.iloc[:,0:2]
filt1 = (df1['Date1'] >= pd.to_datetime('2016-08-10'))
df1 = df1.loc[filt1].set_index("Date1")
df1["Treasuries"] = df1.pct_change().fillna(0)

df2 = df.iloc[:,2:4]
filt2 = (df2['Date2'] >= pd.to_datetime('2016-08-10'))
df2 = df2.loc[filt2].set_index("Date2")
df2["Commodities"] = df2.pct_change().fillna(0)

df3 = df.iloc[:,4:6]
filt3 = (df3['Date3'] >= pd.to_datetime('2016-08-10'))
df3 = df3.loc[filt3].set_index("Date3")
df3["Equities"] = df3.pct_change().fillna(0)

df4 = df.iloc[:,6:8]
filt4 = (df4['Date4'] >= pd.to_datetime('2016-08-10'))
df4 = df4.loc[filt4].set_index("Date4")
```

```
df5 = df.iloc[:,8:10]
filt5 = (df5['Date5'] >= pd.to_datetime('2016-08-10'))
df5 = df5.loc[filt5].dropna().set_index("Date5")
df5["Gold"] = df5.pct_change().fillna(0)

df6 = df.iloc[:,10:12]
filt6 = (df6['Date6'] >= pd.to_datetime('2016-08-10'))
df6 = df6.loc[filt6].dropna().set_index("Date6")
df6["US-IG"] = df6.pct_change().fillna(0)

df7 = df.iloc[:,12:14]
filt7 = (df7['Date7'] >= pd.to_datetime('2016-08-10'))
df7 = df7.loc[filt7].dropna().set_index("Date7")
df7["US-HY"] = df7.pct_change().fillna(0)

df8 = df.iloc[:,14:16]
filt8 = (df8['Date8'] >= pd.to_datetime('2016-08-10'))
df8 = df8.loc[filt8].dropna().set_index("Date8")
df8["US-Agency MBS"] = df8.pct_change().fillna(0)

df9 = df.iloc[:,16:18]
filt9 = (df9['Date9'] >= pd.to_datetime('2016-08-10'))
df9 = df9.loc[filt9].dropna().set_index("Date9")
df9["VIX"] = df9.pct_change().fillna(0)

df10 = df.iloc[:,18:20]
filt10 = (df10['Date10'] >= pd.to_datetime('2016-08-10'))
df10 = df10.loc[filt10].dropna().set_index("Date10")
rfr = (df10.mean())/100

df_final = pd.concat([df1,df2,df3,df4,df5,df6,df7,df8], axis=1)
df_final
```

```
#Plotting annual returns

df_final_indices = df_final.iloc[:,[0,2,4,6,8,10,12,14]]
df_final_returns = df_final.iloc[:,[1,3,5,7,9,11,13,15]]
df_final_returns

annual_returns = (df_final_returns.mean() * 252)
fig = plt.figure(figsize=(15,5))
ax =plt.axes()
ax.bar(annual_returns.index, annual_returns*100, color='royalblue', alpha=0.8)
ax.set_title('Annualized Returns (in %)');

#Plotting annual excess returns
asset_names = ['Treasuries', 'Commodities', 'Equities', 'TIPS', 'Gold', 'US-IG', 'US-HY', 'US-Agency MBS']
df_rf = pd.DataFrame(rfr[0], columns=['Excess Returns'], index= asset_names)
df_annual_returns = pd.DataFrame(annual_returns, columns=['Excess Returns'], index= asset_names)
df_excess_returns = df_annual_returns.subtract(df_rf)

fig = plt.figure(figsize=(15,5))
ax =plt.axes()
ax.bar(df_excess_returns.index, df_excess_returns['Excess Returns']*100, color='royalblue', alpha=0.8)
ax.set_title('Annualized Excess Returns (in %)');

#Plotting annual vols
vols = df_final_returns.std()
annual_vols = vols*sqrt(252)
```

```
fig = plt.figure(figsize=(15,5))
ax = plt.axes()
ax.bar(annual_vols.index, annual_vols*100, color='red', alpha=0.8)
ax.set_title('Annualized Volatility (in %)');

#Plotting Sharpe ratios
excess_ret = df_excess_returns.to_numpy(dtype ='float32')
an_vols = np.array(annual_vols)[:, newaxis]
sharpe_ratios = np.divide(excess_ret, an_vols)
df_sharpe_ratios = pd.DataFrame(sharpe_ratios, columns=['Sharpe Ratio'], index= asset_names)

fig = plt.figure(figsize=(15,5))
ax = plt.axes()
ax.bar(asset_names, df_sharpe_ratios['Sharpe Ratio'], color='green', alpha=0.8)
ax.set_title('Sharpe ratios');
```

Formula list for Portfolio stats

Annualized returns for asset classes = Daily mean return for asset class * 252

Annualized excess returns for asset classes = Annualized return – Mean risk-free rate (3M T-bill, 1%)

Annualized vol for asset classes = Daily standard deviation for asset class * sqrt(252)

Sharpe Ratio = Annualized excess return/Annualized vol

Table for historical Portfolio stats

	Excess Returns	Annual vols	Sharpe Ratio
Treasuries	0.029609	0.131151	0.225763
Commodities	0.070719	0.197911	0.357329
Equities	0.115004	0.155616	0.739025
TIPS	0.065064	0.133593	0.487030
Gold	0.048911	0.131360	0.372340
US-IG	0.036258	0.051932	0.698177
US-HY	0.055677	0.052355	1.063463
US-Agency MBS	0.011937	0.022960	0.519913

Naïve sample covariance

Naïve sample covariance between two asset classes is calculated by analyzing at-return surprises (standard deviations from the excess return) or by multiplying the correlation between the two asset classes by the standard deviation of each asset class.

$Cov(x,y) = \sum [(x_i - x_m) * (y_i - y_m)] / (n - 1)$;

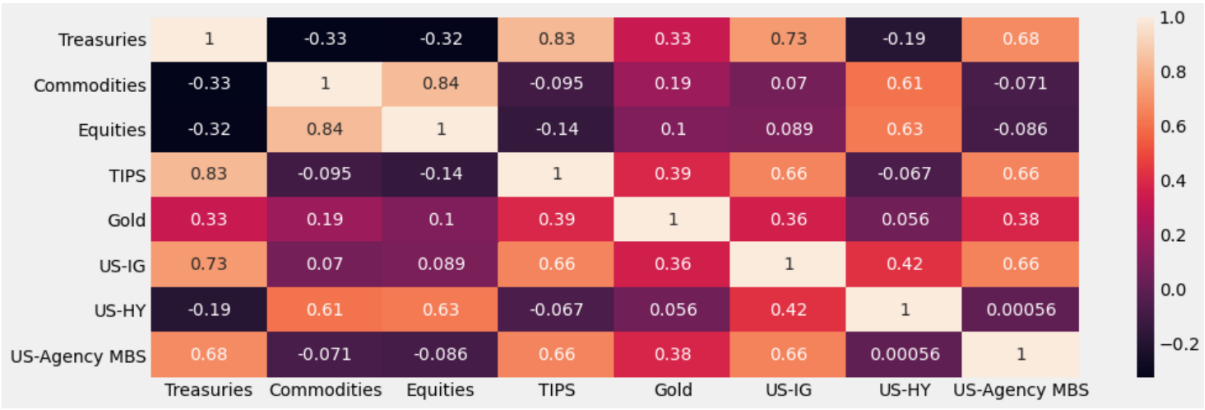
In matrix form- $Cov = S^T RS$, where S is standard deviation diagonal matrix and R is correlation matrix.

Correlation = $Cov(x,y)/((\sigma (x)* \sigma (y))$

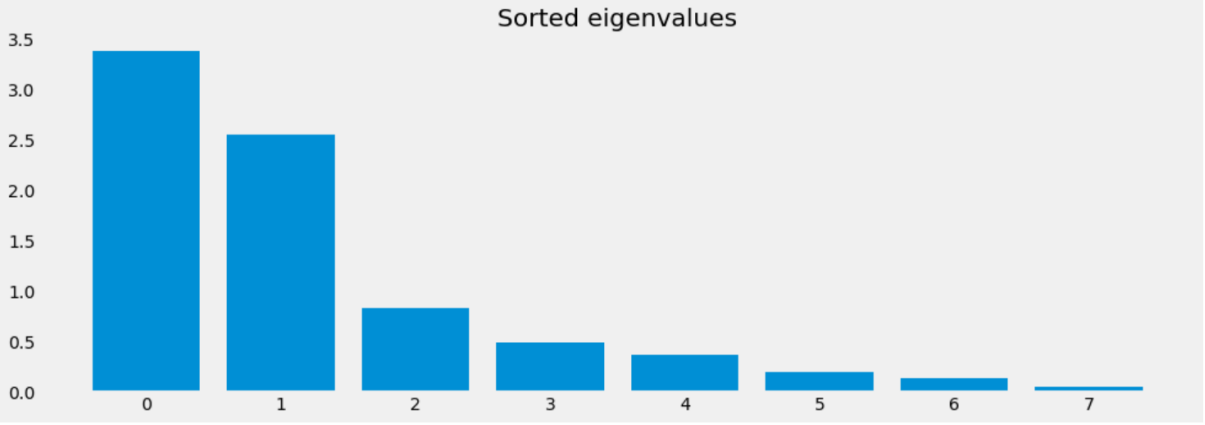
Covariance matrix calculated from daily returns data is multiplied by 252 to get the annualized covariance matrix. Heatmap for naïve covariance matrix and correlation matrix are shown below.

Naïve Covariance Matrix Heatmap

Correlation Matrix Heatmap



Sorted Eigenvalues of naïve correlation matrix



In order to make covariance matrix robust, we use Marchenko-Pastur Denoising technique, which is appplied by replacing the noisy eigenvalues (below the Marchenko-Pastur threshold which is 0.5 in this case) with their average. This has two effects: 1) The smallest eigenvalue becomes bigger. Less noise in matrix inversion. 2) The largest "signal" eigenvalues remain untouched, three largest eigenvalues in this case. Denoising technique is superior than Ledoit-Wolf nonlinear shrinkage in terms of preservation of signal carried by top eigenvectors. We chose to keep top 3 eigenvalues as is and averaged the bottom five eigenvalues to arrive at denoised correlation and covariance matrices.

Mathematics of Marchenko-Pastur Denoising technique

Often portfolio optimization formula uses the INVERSE of the covariance matrix. Inverting a matrix can be numerical unstable if the ratio between the largest and smallest eigenvalue is big (condition number). This will lead to unstable solutions, that is, solutions where a small change in the inputs will cause extreme changes in $w_{opt.}$ The optimal weights are calculated using below formula-

$w_{opt} = Cov^{-1}\ 1/(1^T\ Cov^{-1}\ 1)$

The Constant Residual Eigenvalue method for Marchenko-Pastur Denoising

Let λ_n n=1,…,N be the set of all eigenvalues, ordered descending, and i be the position of the eigenvalue such that $\lambda_i > \lambda_+$ and $\lambda_{i+1} \leq \lambda_+$. Then we set $\lambda_j = (1/N-i)\ \mathrm{Sum}\ [\lambda_k]$; k= i+1 to N, $j = i + 1, \ldots, N$, hence preserving the trace of the correlation matrix.

Given the eigenvector decomposition $VW = W\Lambda$, we form the de-noised correlation matrix C_2 as

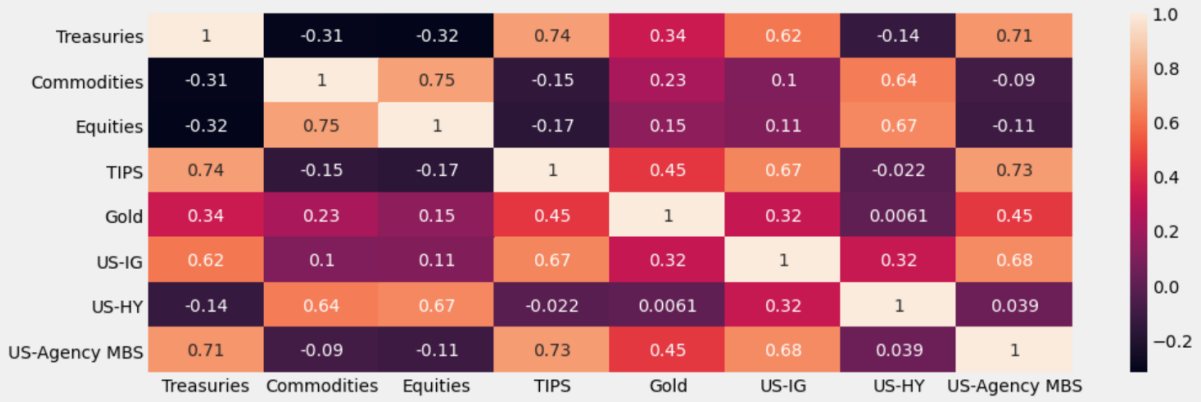
$C_1 = W\Lambda\sim W'$

$C_2 = C_1\ [\ (\mathrm{diag}\ [C_1])^{1/2}\ (\mathrm{diag}\ [C_1])^{1/2'}]^{-1}$

where $\Lambda\sim$ is the diagonal matrix holding the corrected eigenvalues. The reason for the second transformation is to re-scale the matrix C_1, so that the main diagonal of C_2 is an array of 1s.

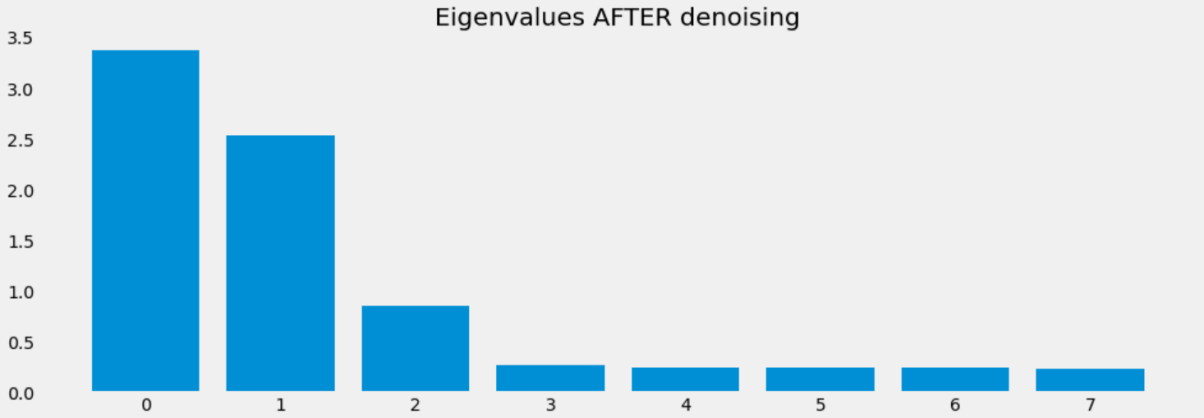
Robust Covariance Matrix Heatmap

Robust Correlation Matrix Heatmap-



Sorted Eigenvalues of robust correlation matrix

As we can see in the chart below, after denoising we have replaced all noisy eigenvalues (bottom five eigenvalues which are less than threshold level of 0.5) with their average, and thus making the correlation/covariance matrix robust.



Python code for naïve and denoised covariance calculation

```
#Naive sample covariance calculation
cov = df_final_returns.cov() * 252

#Heatmap of naive covariance
plt.figure(figsize = (15,5))
ax = sn.heatmap(cov, annot=True)

#Naive sample covariance calculation
corr = df_final_returns.corr()
df_corr = pd.DataFrame(corr, index= asset_names,
                  columns= asset_names)

#Heatmap of naive sample correlation
plt.figure(figsize = (15,5))
ax = sn.heatmap(df_corr, annot=True)

#Eigenvalues calc for naive covariance
eVal, eVec = np.linalg.eigh(corr)
indices = eVal.argsort()[::-1]  # arguments for sorting eVal desc
eVal, eVec = eVal[indices], eVec[:, indices]
eVal
```

```
#Sorted Eigenvalues
plt.figure(figsize = (15,5))
plt.bar(np.arange(len(eVal)),eVal)
plt.title('Sorted eigenvalues')
plt.grid()
plt.show()

#Function for calculation of covariance and correlation using each other
def cov2corr(cov):
    std = np.sqrt(np.diag(cov))
    corr = cov/np.outer(std,std)
    corr[corr<-1] = -1
    corr[corr>1] = 1
    return corr

def corr2cov(corr, std):
    cov=corr*np.outer(std,std)
    return cov
```

```
#Function for Marcenko-Pastur denoised correlation calc
nFacts = np.sum(eVal > 0.5)
def denoisedCorr(eVal, eVec, nFacts):
    eVal_ = eVal.copy()
    eVal_[nFacts:] = eVal_[nFacts:].sum() / float(eVal_.shape[0]-nFacts)
    eVal_ = np.diag(eVal_)
    corr = np.dot(eVec,eVal_).dot(eVec.T)
    corr = cov2corr(corr)
    return corr
cor_d = denoisedCorr(eVal, eVec, nFacts)
df_cor = pd.DataFrame(cor_d, index= asset_names,
                  columns= asset_names)

#heatmap for Marcenko-Pastur denoised correlation matrix
plt.figure(figsize = (15,5))
ax = sn.heatmap(df_cor, annot=True)

#heatmap for Marcenko-Pastur denoised covariance matrix
cov_d = corr2cov(cor_d, np.diag(cov)**.5)
df_cov = pd.DataFrame(cov_d, index= asset_names,
                  columns= asset_names)
plt.figure(figsize = (15,5))
ax = sn.heatmap(df_cov, annot=True)
```

```
#Eigenvalues calc for robust denoised covariance
eVal_d, eVec_d = np.linalg.eigh(cor_d)
indices = eVal_d.argsort()[::-1]  # arguments for sorting eVal desc
eVal_d, eVec_d = eVal_d[indices], eVec_d[:, indices]
eVal_d

#Sorted Eigenvalues post denoising
plt.figure(figsize = (15,5))
plt.bar(np.arange(len(eVal_d)),eVal_d)
plt.title('Eigenvalues AFTER denoising')
plt.grid()
plt.show()
```

Constrained Mean variance optimization using naïve vs robust covariance with market implied λ

We consider constrained mean variance optimization using market implied lamda to compare allocations using naïve vs robust covariance. The table below provides the index weights of five original asset classes of RPAR ETF. In order to make things not more complicated we remove the 20% leverage weight from the portfolio by reducing 4% weight of each 5 asset classes.

	6/30/21 Allocation	Long-Term Target Allocation
RPAR Risk Parity ETF	**119.9%**	**120.0%**
Global Equities	25.0%	25.0%
Commodity Producers	14.9%	15.0%
Physical Gold	9.2%	10.0%
Treasuries	35.1%	35.0%
TIPS	35.8%	35.0%

In order to get the index weights after addition of 3 new asset classes (US-IG, US-HY, and Agency MBS), we rotated 20% of existing fixed income allocation from Treasuries and TIPS (after removal of 4% weight for leverage, 31% weight each) into new asset classes based on their total market cap.

The table below shows the total market cap in USD trillions for different fixed income products.

Fixed income product	Market cap ($T)	% of total
Treasuries and TIPS	17	38%
Corporate bonds	9.8	22%
Agency MBS	7	16%
Others	10.6	24%

Since, we are shifting 20% of existing Treasuries and TIPS allocation to three additional asset classes in the expanded portfolio, in the final index weighing we remove 10% weight each from Treasuries and TIPS and allocate that weight to 12% in corporate bonds (6% each for US-IG and US-HY) and 8% in Agency MBS. The final index or benchmark weights to be used in market implied lambda calculation are shown below in the table.

	Index weights
Treasuries	0.21
Commodities	0.11
Equities	0.21
TIPS	0.21
Gold	0.06
US-IG	0.06
US-HY	0.06
US-Agency MBS	0.08

Market Implied lambda (risk aversion coefficient) calculation-

We use final index weights (calculated above) to get benchmark excess returns. Further, dividing benchmark excess return by the square of benchmark standard deviation gives us a market implied lambda of 12.1, which we use in mean variance optimization.

Benchmark excess returns = (Excess returns)^T dot (Index weights)

Benchmark excess returns covariance = Naïve covariance OR robust covariance

Benchmark standard deviation = (Index weights)^T dot (Benchmark covariance dot Index weights)

Lambda (Risk aversion coefficient) = Benchmark excess return/ (Benchmark standard deviation) ^2

OR Benchmark Sharpe Ratio/Benchmark standard deviation

Mathematics of Mean-Variance Optimization-

In order to achieve mean-variance optimization we need to optimize (maximize) the following unconstrained maximization problem for w-

Max $w^T \mu - (\lambda/2)\, w^T \sum w$

In matrix form, $\sum = S^T R\, S$, dot product of three matrices, where S is standard deviation diagonal matrix ($S = S^T$) and R is correlation matrix.

The first part of the above equation is portfolio return and the second part of the equation is portfolio standard deviation scaled by a factor of lambda/2. In case of constrained optimization, we are using following two constraints-

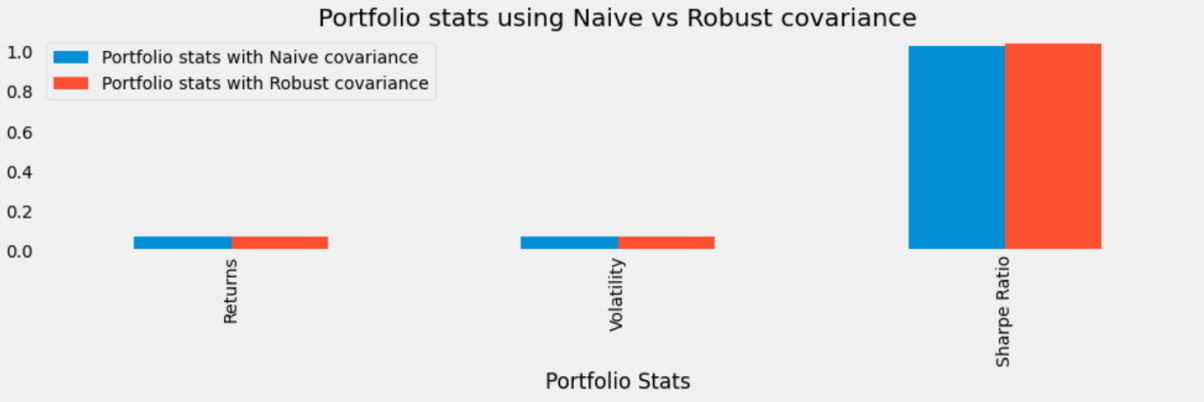
1) The sum of weights of each asset class should be 1 (no leverage in portfolio)
2) The weight range for each asset class should be (0.05, 0.3). This constraint helps us in achieving an allocation with a minimum representation from each asset class (5%) which helps in inclusion of each asset in the final allocation that makes the portfolio diversified and

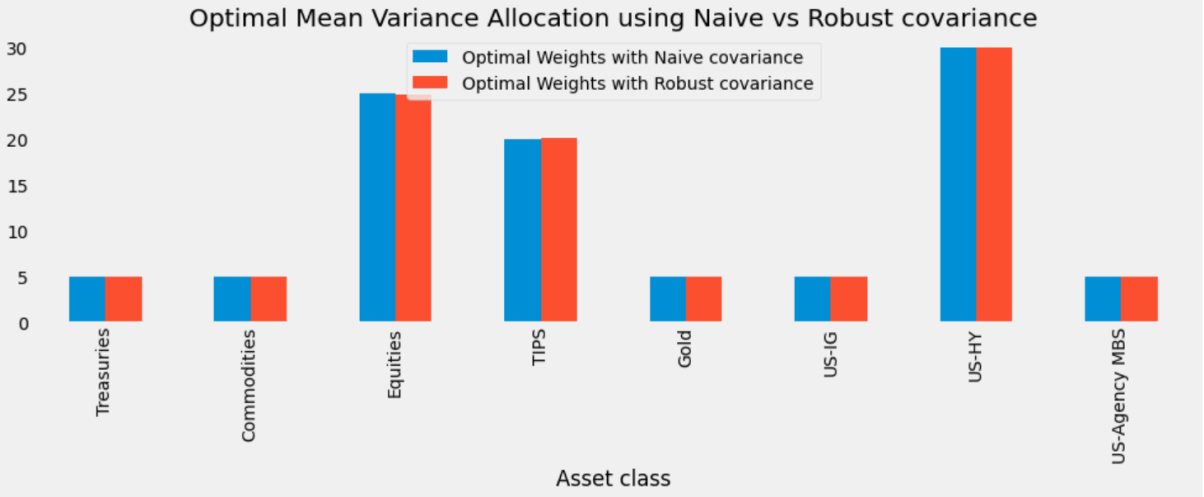
consistent under various economic environments. The upper cap of 30% weight for each asset class helps avoiding portfolio allocations with concentrated positions in one particular asset class or corner solutions for optimizations. Also, this constraint also implies no short positions are allowed in final allocations.

We use scipy.optimize function from Scipy library to get optimized weights with constraints and bounds (constraints and bounds can be directly fed in the scipy.optimize function). Since we are using the scipy.optimize.minimize function, we will change the sign of function which needs to be optimized.

Below charts show the calculated optimal portfolio weights using constrained mean variance optimized with market implied lambda of 12.1 for naïve covariance vs denoised robust covariance. As we can see in the chart, there is a very insignificant difference between naïve and robust covariance optimal allocations (0.12% difference between Equities and TIPS allocations) and subsequent portfolio stats. However, the calculated allocation using robust covariance will be more stable, by definition, thus reducing the large fluctuations in optimized weights due to small changes in inputs like covariance/correlation and excess returns matrices.

Mean-Variance Optimization charts and tables using Naïve vs Robust covariance –





Portfolio Stats	Portfolio stats with Naive covariance	Portfolio stats with Robust covariance
Returns	0.07	0.07
Volatility	0.07	0.07
Sharpe Ratio	1.03	1.04

Asset class	Optimal Weights with Naive covariance	Optimal Weights with Robust covariance
Treasuries	5.00	5.00
Commodities	5.00	5.00
Equities	25.01	24.89
TIPS	19.99	20.11
Gold	5.00	5.00
US-IG	5.00	5.00
US-HY	30.00	30.00
US-Agency MBS	5.00	5.00

Python code for constrained mean variance optimization (non Black-Litterman)

```
#Constrained Mean-Variance Optimization

num_assets = len(asset_names)
weights = random.random(num_assets)[:, newaxis]
lamda = 12.1
excess_ret = df_excess_returns.to_numpy(dtype ='float32')
covar = cov

#function to be used for optimisation
def portfolio_stats_NaiveMV(weights):

    port_rets = np.dot(weights.T, excess_ret)
    port_vols = sqrt(multi_dot([weights.T, (lamda/2)*covar, weights]))

    return np.array([port_rets, port_vols, port_rets/port_vols]).flatten()

#function which will be optimized

def mean_variance_constrained(weights):
    return (portfolio_stats_NaiveMV(weights)[1]**2) - portfolio_stats_NaiveMV(weights)[0]

cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
bnds = tuple((0.05, 0.3) for x in range(num_assets))
initial_wts = num_assets*[1/num_assets]
```

```
# Optimizing for naive mean variance
opt_mean_var_naive = sco.minimize(mean_variance_constrained, initial_wts, method='SLSQP', bounds=bnds, constraints=cons)

# Portfolio weights
list(zip(asset_names,np.around(opt_mean_var_naive['x']*100,2)))

df_a = pd.DataFrame(np.around(opt_mean_var_naive['x']*100,2), index = asset_names, columns =['Optimal Weights with Naive covariar
df_a

#function to calculate final optimised portfolio stats
def portfolio_stat_(weights):

    port_rets = np.dot(weights.T, excess_ret)
    port_vols = sqrt(multi_dot([weights.T, covar, weights]))

    return np.array([port_rets, port_vols, port_rets/port_vols]).flatten()

# Portfolio stats
stats = ['Returns', 'Volatility', 'Sharpe Ratio']
list(zip(stats, portfolio_stat_(opt_mean_var_naive['x'])))
portfolio_stat_(opt_mean_var_naive['x'])

df_b = pd.DataFrame(np.around(np.vstack(portfolio_stat_(opt_mean_var_naive['x'])),2), index = stats, columns =['Portfolio stats
df_b
```

Black-Litterman Model

The Black-Litterman model enables investors to combine their unique views regarding the performance of various assets with the market equilibrium in a manner that results in intuitive, diversified portfolios. The Black-Litterman model uses a Bayesian approach to combine the subjective views of an investor regarding the expected returns of one or more assets with the market equilibrium vector of expected returns (the prior distribution) to form a new, mixed estimate of expected returns. We have applied Black-Litterman model for four scenarios where we are using robust Marchenko-Pastur denoised covariance, market/benchmark weights implied risk aversion coefficient, and two investor views to calculated posterior BL excess returns vector, posterior covariances matrix, and optimal BL weights. The four different scenarios used are as follows-

1) Unconstrained Mean-Variance Optimizer
2) Constrained Mean-Variance Optimizer
3) Constrained Minimum Variance Optimizer
4) Constrained Maximum Sharpe Optimizer

The two subjective views we incorporated in the BL model are as follows-

1) Treasuries will underperform Equities by 5%. The rationale behind this view is that current levels of long-term treasury yields are quite low, considering the strong economic data prints we are getting from labour and inflation market. We think that few technical factors are causing the long-term treasury yields to remain low currently, and these factors will fade away going forward, thus causing treasuries to underperform relative to equities. We should see the relatively higher allocation to equity vs treasuries for the optimized weights vs benchmark weights.
2) Gold will have an absolute excess return of 10% going forward. This view is counterintuitive to first view, considering treasuries and Gold pair has a correlation of 0.33 historically. But, we want to add a counter view in the portfolio to assess the performance of risk parity portfolio in a situation where the historical correlation flips. We should see a higher allocation to Gold in BL optimized weights due to this view vs benchmark weight of Gold since Gold's benchmark excess return is close to 5%.

Reverse optimization

The Black-Litterman model uses "equilibrium" returns as a neutral starting point. Equilibrium returns are the set of returns that clear the market. The equilibrium returns are derived using a reverse optimization method in which the vector of implied excess equilibrium returns is extracted from known information using following formula-

$\prod = \lambda \sum w_{mkt}$

$\prod$ is the Implied Excess Equilibrium Return Vector (N x 1 column vector);

λ is the risk aversion coefficient, calculated by the formula-

λ = Benchmark excess return/ (Benchmark standard deviation) ^2;

Benchmark excess return = (Excess returns vector)^T dot (Equilibrium weights vector);

Benchmark standard deviation = $(w_{mkt})^T$ dot ((robust covariance matrix) dot (w_{mkt}))

$\sum$ is the covariance matrix of excess returns (N x N matrix) (Robust covariance matrix); and,

w_{mkt} is the market capitalization weight (N x 1 column vector) of the assets.

Black Litterman Formula

The formula for the new Combined Return Vector (E[R]), in other words, posterior excess returns is-

$E[R] = [(\tau\sum)^{-1} + P^T \Omega^{-1} P]^{-1} [(\tau\sum)^{-1} \prod + P^T \Omega^{-1} Q]$

E[R] is the new (posterior) Combined Return Vector (N x 1 column vector);

τ is a scalar whose value ranges between 0.01 and 0.05, we have taken a value of 0.025;

$\sum$ 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);

$\prod$ is the Implied Equilibrium Return Vector (N x 1 column vector); and,

Q is the View Vector (K x 1 column vector).

Our two views can be expressed as two arrays which can be combined together to form vector P-

View 1 = [-1, 0, 1, 0, 0, 0, 0, 0]; Relative view (Treasuries vs Equities)

View 2 = [0, 0, 0, 0, 1, 0, 0, 0]; Absolute view (Gold)

Q = [0.05, 0.10]; Magnitude of views

Ω is calculated using following formula-

diag [diag [P ($\tau\sum$) P^T]]

The formula for posterior covariance matrix is given below-

$\sum + [(\tau\sum)^{-1} + P^T \Omega^{-1} P]^{-1}$

Once we have the posterior returns and posterior covariance calculated, we can use the following formula for the calculation of optimized weights using BL mean variance unconstrained optimization by optimizing the equation: Max $w^T \mu - (\lambda/2)\, w^T \sum w$

Black Litterman weights = $w_{BL} = (\lambda \sum)^{-1} \mu$

where μ is posterior returns, and $\sum$ is robust covariance matrix.

Constrained Optimization using different optimizers

To calculate the optimal weights for constrained optimization (with constraints of sum of weights should be 1 and all weights should be in a range of (0.05, 0.3)) using Mean-variance, Minimum variance, and Maximum Sharpe optimizers, we use scipy.optimize.minimize formula from SciPy library. We minimize following functions in scipy.optimize.minimize formula for different optimizers-

Mean Variance optimizer: $(\lambda/2)\, w^T \sum w - w^T \mu$

Minimum Variance optimizer: $w^T \sum w$

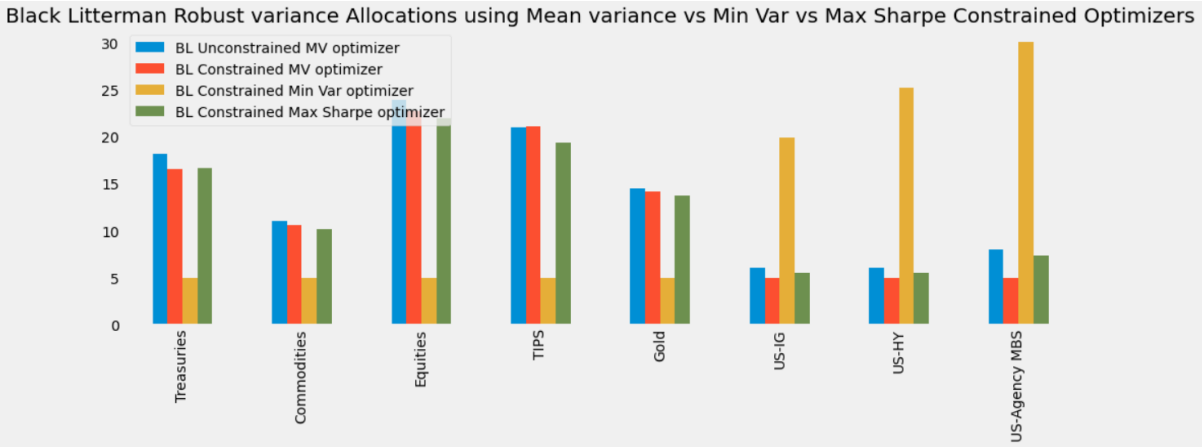
Maximum Sharpe optimizer: $w^T \mu / (w^T \sum w)$

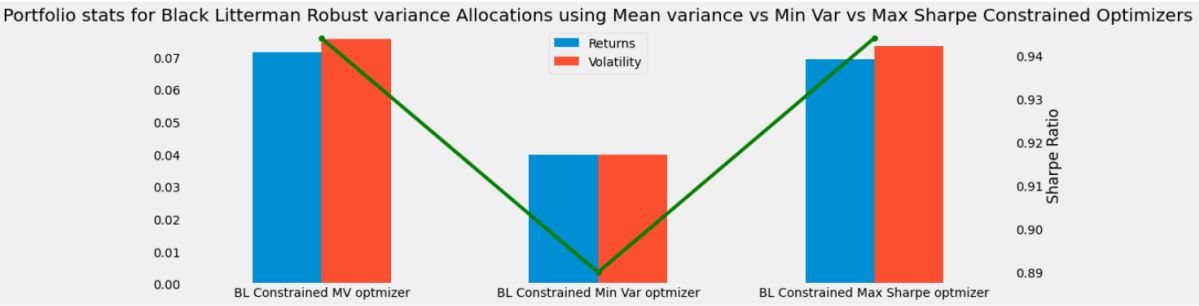
Here $\sum$ and μ represent posterior BL portfolio covariance and new combined returns vector.

We can also specify the constraint and weights bounds in scipy.optimize.minimize formula itself.

The optimal portfolio allocation and related portfolio stats (returns, vol, sharpe ratio) using market implied lambda and posterior BL returns and covariance matrix as inputs to various optimizers are shown below.

Charts and tables for optimal allocation and portfolio stats using various optimizers in BL





	BL Unconstrained MV optimizer, λ=12.1	BL Constrained MV optimizer, λ=12.1	BL Constrained Min Var optimizer	BL Constrained Max Sharpe optimizer
Treasuries	18.15	16.48	5.00	16.60
Commodities	11.00	10.61	5.00	10.12
Equities	23.85	22.69	5.00	21.93
TIPS	21.00	21.06	5.00	19.29
Gold	14.44	14.17	5.00	13.71
US-IG	6.00	5.00	19.85	5.50
US-HY	6.00	5.00	25.15	5.52
US-Agency MBS	8.00	5.00	30.00	7.31

	Returns	Volatility	Sharpe Ratio
BL Constrained MV optmizer, λ=12.1	0.0716	0.0758	0.9439
BL Constrained Min Var optmizer	0.0400	0.0400	0.8900
BL Constrained Max Sharpe optmizer	0.0693	0.0734	0.9440

Above results make intuitive sense, few noteworthy points to mention here are-

1) The optimal Black Litterman weights have gone down for Treasuries (below 21%) and went up for equities (above 21%) vs their benchmark allocation (market capitalisation weight) of 21% for all optimizers due to View 1 incorporated in BL model.
2) The optimal Black Litterman weight for Gold has significantly gone up (around 15%) vs market capitalisation weight of 6% due to view 2 incorporated in BL model.
3) The optimal Black Litterman weights for Minimum Variance optimizer look different than rest of the optimizers because the objective of min var optimizer is to minimize the variance of portfolio, which is why, for min var case, portfolio is heavily allocated to low volatile assets like US-IG, US-HY, and Agency MBS.
4) In terms of portfolio stats, risk, return, and sharpe ratio profile for mean variance and maximum sharpe optimizers look quite similar, as expected, whereas min var portfolio has a lower sharpe ratio as min var focuses more on minimizing the variance than risk-adjusted returns.

5) Unconstrained and constrained mean variance optimizers allocations are very similar, so does their portfolio stats with unconstrained sharpe ratio being only very slightly higher than constrained mean variance optimization.

Python code for unconstrained mean variance optimization using Black-Litterman inputs

```
#Black Litterman Unconstrained Mean Variance Optimization
num_assets = len(asset_names)
data = {'Index weights':[0.21, 0.11, 0.21, 0.21, 0.06, 0.06, 0.06, 0.08]}
df_eq_wts = pd.DataFrame(data, index = asset_names)
df_eq_wts

Index_weights = [0.21, 0.11, 0.21, 0.21, 0.06, 0.06, 0.06, 0.08]
Eq_wts = arr.array('d', Index_weights)
excess_ret = df_excess_returns.to_numpy(dtype ='float32')
excess_ret_mkt = np.dot(excess_ret.T,Eq_wts)
covar = cov_d
Eq_wts = array(Eq_wts)[:,newaxis]
sigma_mkt = sqrt(multi_dot([Eq_wts.T, covar, Eq_wts]))
sharpe_ratio = excess_ret_mkt/sigma_mkt
tau = 0.025


def Black_Litterman(covar,sharpe_ratio,tau):

    #Index (Equilibrium) weights for asset classes
    Index_weights = [0.21, 0.11, 0.21, 0.21, 0.06, 0.06, 0.06, 0.08]
    Eq_wts = arr.array('d', Index_weights)
    Eq_wts = array(Eq_wts)[:,newaxis]
```

```
    #Robust covariance matrix for excess returns
    cov_d_inv = np.linalg.inv(covar)
    sigma_mkt = sqrt(multi_dot([Eq_wts.T, covar, Eq_wts]))

    #calculation of lambda (Risk aversion coefficient)
    lamda = (sharpe_ratio)/(sigma_mkt)

    #Reverse optimization
    Eq_risk_premium = lamda*np.dot(covar, Eq_wts)
    num_datapoints = len(df_final_returns.axes[0])
    #tau = 1/num_datapoints

    #covariance of prior distribution
    tau_cov_inv = np.linalg.inv(tau*covar)

    #Investor views:
    # View 1) Treasuries will underperform equities by 5%;
    # View 2) Gold will have an absolute excess return of 10%
    View1 = [-1, 0, 1, 0, 0, 0, 0, 0]
    View2 = [0, 0, 0, 0, 1, 0, 0, 0]
    Qvec = [0.05, 0.10]

    P_vec_list = [View1, View2]
    P_vec = np.array(P_vec_list)

    Q_vec = arr.array('d', Qvec)
    Q_vec = array(Q_vec)[:,newaxis]
```

```
    #Matrix P.tau.covar.Ptranspose
    Matrix = multi_dot([P_vec, tau*covar, P_vec.T])

    Omega = np.diag(np.diag(Matrix))
    Omega_inv = np.linalg.inv(Omega)

    Matrix_A = tau_cov_inv + multi_dot([P_vec.T, Omega_inv, P_vec])
    Matrix_B = np.dot(tau_cov_inv, Eq_risk_premium) + multi_dot([P_vec.T, Omega_inv, Q_vec])

    #Expected return and covariance for posterior distribution
    ret_posterior = np.dot(np.linalg.inv(Matrix_A), Matrix_B)
    cov_posterior = covar + np.linalg.inv(tau_cov_inv + multi_dot([P_vec.T, Omega_inv, P_vec]))

    #Asset allocation with unconstrained optimization
    BL_weights =  np.dot((1/lamda)*cov_d_inv, ret_posterior)

    return np.array([lamda, sharpe_ratio, ret_posterior, cov_posterior, BL_weights]).flatten()

# Portfolio weights
list(zip(asset_names,np.around(Black_Litterman(covar,sharpe_ratio,tau)[4]*100,2)))

df_e = pd.DataFrame(np.around(Black_Litterman(covar,sharpe_ratio,tau)[4]*100,2), index = asset_names, columns =['BL Unconstraine
df_e

weights = Black_Litterman(covar,sharpe_ratio,tau)[4]
```

```
#function to calculate final optimised portfolio stats
def portfolio_stats_(weights):

    port_rets = np.dot(weights.T, Black_Litterman(covar,sharpe_ratio,tau)[2])
    port_vols = sqrt(multi_dot([weights.T, Black_Litterman(covar,sharpe_ratio,tau)[3], weights]))

    return np.array([port_rets, port_vols, port_rets/port_vols]).flatten()

portfolio_stats_(weights)

df_m = pd.DataFrame(np.around(portfolio_stats_(weights),4), index = stats, columns =['BL Unconstrained MV optmizer, λ=12.1'])
df_m
```

Python code for constrained mean variance optimization using Black-Litterman inputs

```
#Black Litterman using constrained Mean Variance Optimization

weights = random.random(num_assets)[:, newaxis]
covar = cov_d

#function to be used for optimisation
def portfolio_stats_MV(weights):

    port_rets = np.dot(weights.T, Black_Litterman(covar,sharpe_ratio,tau)[2])
    port_vols = sqrt(multi_dot([weights.T, (Black_Litterman(covar,sharpe_ratio,tau)[0]/2)*Black_Litterman(covar,sharpe_ratio,tau)

    return np.array([port_rets, port_vols, port_rets/port_vols]).flatten()

#function which will be optimized

def mean_variance_constrained(weights):
    return (portfolio_stats_MV(weights)[1]**2) - portfolio_stats_MV(weights)[0]

cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
bnds = tuple((0.05, 0.3) for x in range(num_assets))
initial_wts = num_assets*[1/num_assets]

# Optimizing for constrained mean variance
opt_mean_var = sco.minimize(mean_variance_constrained, initial_wts, method='SLSQP', bounds=bnds, constraints=cons)
```

```
# Portfolio weights
list(zip(asset_names,np.around(opt_mean_var['x']*100,2)))

df_f = pd.DataFrame(np.around(opt_mean_var['x']*100,2), index = asset_names, columns =['BL Constrained MV optimizer, λ=12.1'])
df_f

# Portfolio stats
stats = ['Returns', 'Volatility', 'Sharpe Ratio']
list(zip(stats, portfolio_stats_(opt_mean_var['x'])))

df_g = pd.DataFrame(np.around(np.vstack(portfolio_stats_(opt_mean_var['x'])),4), index = stats, columns =['BL Constrained MV optr
df_g
```

Python code for constrained minimum variance optimization using Black-Litterman inputs

```
#Black Litterman using constrained Min Variance Optimization

weights = random.random(num_assets)[:, newaxis]
covar = cov_d

# Minimize Variance
def min_variance(weights):
    return portfolio_stats_(weights)[1]**2

# Each asset boundary ranges from 0 to 1
tuple((0, 1) for x in range(num_assets))

cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
bnds = tuple((0.05, 0.3) for x in range(num_assets))
initial_wts = num_assets*[1/num_assets]

# Optimizing for constrained minimum variance
opt_var = sco.minimize(min_variance, initial_wts, method='SLSQP', bounds=bnds, constraints=cons)

# Portfolio weights
list(zip(asset_names,np.around(opt_var['x']*100,2)))

df_h = pd.DataFrame(np.around(opt_var['x']*100,2), index = asset_names, columns =['BL Constrained Min Var optimizer'])
df_h
```

```
# Portfolio stats
stats = ['Returns', 'Volatility', 'Sharpe Ratio']
list(zip(stats,portfolio_stats_(opt_var['x'])))

df_i = pd.DataFrame(np.around(np.vstack(portfolio_stats_(opt_var['x'])),2), index = stats, columns =['BL Constrained Min Var optr
df_i
```

Python code for constrained maximum sharpe optimization using Black-Litterman inputs

```
#Black Litterman using constrained Max Sharpe Optimization

weights = random.random(num_assets)[:, newaxis]
covar = cov_d

# Maximizing sharpe ratio
def min_sharpe_ratio(weights):
    return -portfolio_stats_(weights)[2]

# Each asset boundary ranges from 0 to 1
tuple((0, 0.3) for x in range(num_assets))

cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
bnds = tuple((0, 2) for x in range(num_assets))
initial_wts = num_assets*[1/num_assets]

# Optimizing for maximum sharpe ratio
opt_sharpe = sco.minimize(min_sharpe_ratio, initial_wts, method='SLSQP', bounds=bnds, constraints=cons)

# Portfolio weights
list(zip(asset_names,np.around(opt_sharpe['x']*100,2)))

df_j = pd.DataFrame(np.around(opt_sharpe['x']*100,2), index = asset_names, columns =['BL Constrained Max Sharpe optimizer'])
df_j
```

```
# Portfolio stats
stats = ['Returns', 'Volatility', 'Sharpe Ratio']
list(zip(stats,portfolio_stats_(opt_sharpe['x'])))

df_k = pd.DataFrame(np.around(np.vstack(portfolio_stats_(opt_sharpe['x'])),4), index = stats, columns =['BL Constrained Max Shar
df_k
```

Unconstrained and Constrained Mean Variance Optimization using different lambdas

Theory of risk aversion-

In 1952, Harry Markowitz laid out the foundation for modern portfolio theory and the mean-variance portfolio selection/optimization process. He reasoned that for a fixed set of asset means, variances, and covariances, investors would want to select a portfolio that lies on an efficient frontier—that is, a portfolio with minimum variance for a given expected return or with maximum expected return for a given variance. Markowitz stopped short of pointing out that the particular portfolio on the efficient frontier an investor chooses depends on his risk aversion, which measures his reluctance to take on additional risk for any additional expected return. Nowadays, risk aversion parameters play an important and explicit role in portfolio management. Quantitative portfolio managers often use a mean-variance objective when constructing and rebalancing their portfolios. In the simplest form, such an objective U takes on the form-

$U = \mu - \lambda \sigma^2$

where μ is the expected portfolio return, σ^2 is the portfolio variance (σ is the standard deviation), and λ ($\lambda \geq 0$) is a risk aversion parameter that specifies the relative importance of the two terms—return and variance (representing risk)—in the objective. The larger the value of λ , the more risk is to be penalized. Conversely, the smaller the value of λ , the less important the risk term will be in the objective. In the extreme case when $\lambda = 0$, the risk term is ignored and the return term completely dominates. In essence, the risk aversion parameter λ represents a trade-off between the expected return and risk.

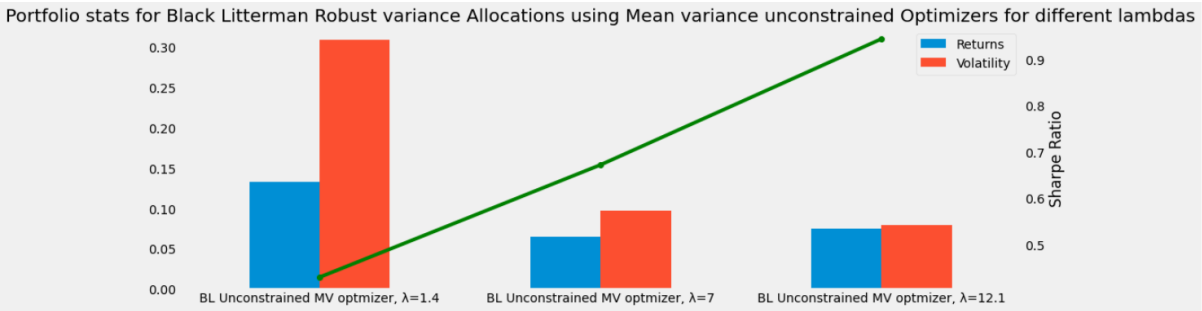
Empirical effects of risk aversion on portfolio optimization-

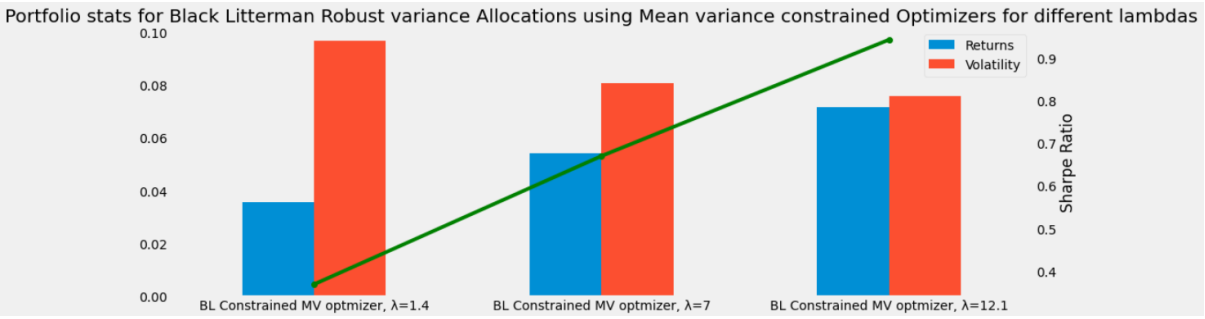
In the absence of complicated constraints or objective terms, the return, risk, and utility of the optimal mean-variance portfolio all decrease as the risk aversion increases. The number of names in a long-only optimal portfolio increases as the risk aversion increases. The Sharpe ratio, on the other hand, first increases and then decreases as the risk aversion increases. In other words, there is a portfolio with a maximum value of the Sharpe ratio when we maximize the risk-adjusted return. However, that portfolio might not be what the portfolio manager would choose.

We have shown below returns, vols, and Sharpe ratios profiles in charts for three different lambdas (λ= 1.4, 7, and 12.1) for Black-Litterman, unconstrained and constrained mean variance optimizers.

It's clear in first chart with unconstrained mean variance optimizer, that as risk aversion coefficient increases, the magnitude of returns and vols decrease. Sharpe ratio is increasing with increasing level of risk aversion, which also matches with the empirical findings mentioned in the writing above. However, in the second chart with constrained mean variance optimization, the relationship is not so clear between returns and risk aversion. The returns are increasing with increasing values of lambdas, which can be explained by the presence of constraints in optimization. Nevertheless, vols and Sharpe ratio are following the empirical and intuitive relationship between risk aversion and vols/Sharpe ratios.

Charts and tables for portfolio stats for unconstrained and constrained MV optimizers using different lambdas-



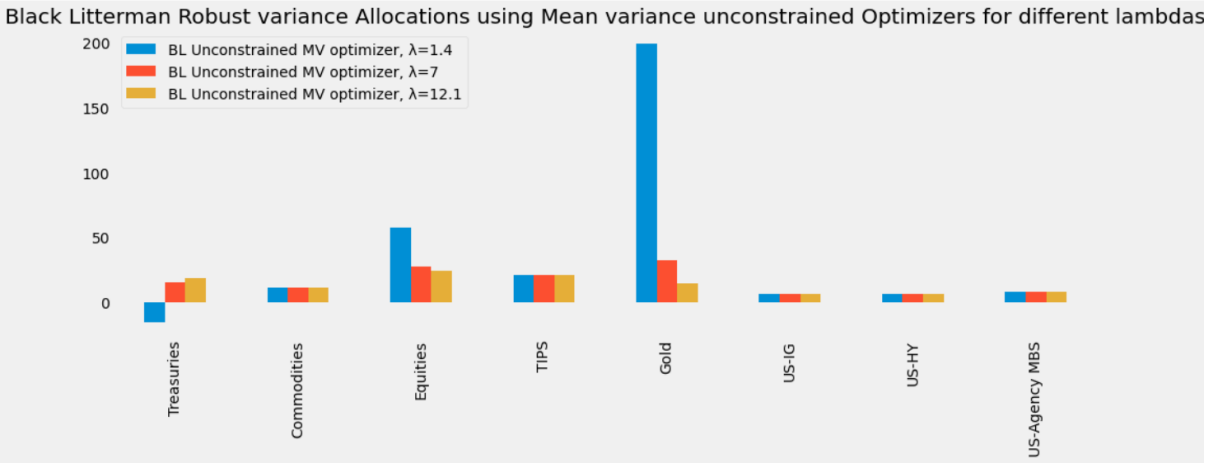


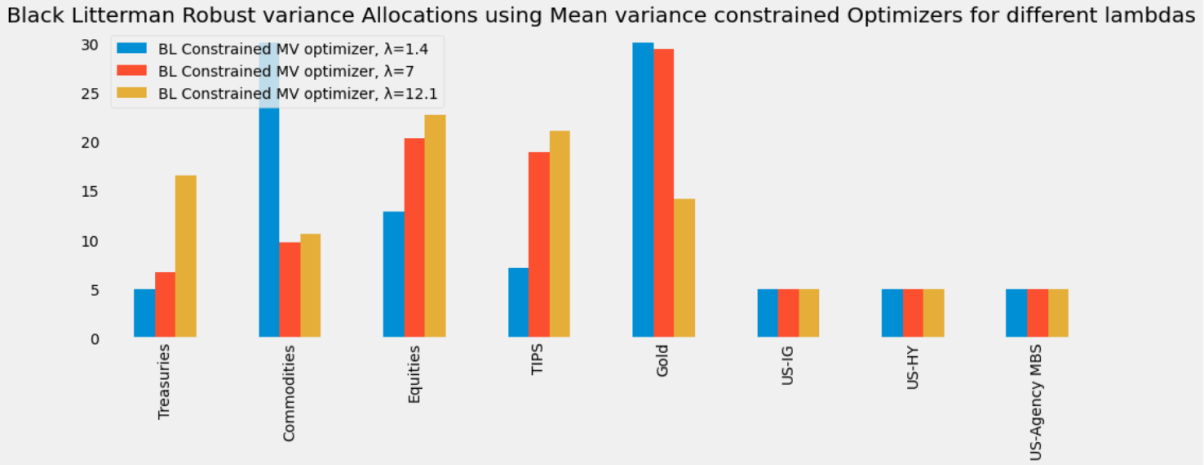
	Returns	Volatility	Sharpe Ratio
BL Unconstrained MV optmizer, λ=1.4	0.1331	0.3097	0.4299
BL Unconstrained MV optmizer, λ=7	0.0654	0.0973	0.6723
BL Unconstrained MV optmizer, λ=12.1	0.0750	0.0795	0.9440

	Returns	Volatility	Sharpe Ratio
BL Constrained MV optmizer, λ=1.4	0.0357	0.0969	0.3688
BL Constrained MV optmizer, λ=7	0.0541	0.0806	0.6705
BL Constrained MV optmizer, λ=12.1	0.0716	0.0758	0.9439

Charts and tables for optimal portfolio allocations for unconstrained and constrained MV optimizers using different lambdas-

Finally, we don't see any clear relationship between risk aversion and number of assets in the optimized unconstrained or constrained portfolio. As despite using unconstrained optimization, we don't see any asset with 0 weight or lower number of assets in portfolio for lower level of risk aversion and vice versa.





	BL Unconstrained MV optimizer, λ=1.4	BL Unconstrained MV optimizer, λ=7	BL Unconstrained MV optimizer, λ=12.1
Treasuries	-15.17	14.99	18.15
Commodities	11.00	11.00	11.00
Equities	57.17	27.01	23.85
TIPS	21.00	21.00	21.00
Gold	199.15	31.96	14.44
US-IG	6.00	6.00	6.00
US-HY	6.00	6.00	6.00
US-Agency MBS	8.00	8.00	8.00

	BL Constrained MV optimizer, λ=1.4	BL Constrained MV optimizer, λ=7	BL Constrained MV optimizer, λ=12.1
Treasuries	5.00	6.71	16.48
Commodities	30.00	9.75	10.61
Equities	12.83	20.26	22.69
TIPS	7.17	18.89	21.06
Gold	30.00	29.40	14.17
US-IG	5.00	5.00	5.00
US-HY	5.00	5.00	5.00
US-Agency MBS	5.00	5.00	5.00

Concluding remarks

Based on running the portfolio allocations and portfolio stats for various scenarios using different optimizers, it is prudent to construct a portfolio using Black-Litterman inputs, which is a great tool to incorporate our own subjective views in portfolio optimization, in the same manner we added our two views in portfolio construction. Using Black-Litterman also makes the optimal allocations more stable and robust in terms of inputs sensitivity (expected returns) to final results as opposed to using theoretical/historical returns, specially in the light of the fact that optimal allocations are very sensitive to expected returns inputs (smaller changes in expected returns lead to large changes in allocations which could be very costly in terms of frequent rebalancing of portfolio). Also, using robust denoised covariance vs naïve covariance has similar effects of stabilizing the optimal allocations. The choice of optimizers should be based on risk aversion of the investor. An investor with high risk aversion can go for constrained minimum var optimizer allocations whereas a relatively neutral or less risk averse investor should invest in allocations suggested by Max Sharpe/Mean Variance optimizers. Mean variance optimizers would be superior to Maximum shape optimizers in terms of taking into account the risk aversion profile of various investors and incorporating it into final optimal allocations. Finally, it is very important to keep the portfolio allocations well balanced across multiple asset classes which perform well under different economic environments. This can be achieved by setting portfolio constraints (weight bounds) and mandating a minimum weight in each asset class as well as capping the maximum weight in any particular asset class to avoid corner solutions/highly concentrated allocations which defeats the underlying rationale of risk parity portfolio which performs well with lesser drawdowns in any economic environment.