we analyzed the performance of our portfolio, built using the historical average method to set return expectations. We calculated return and risk contributions and examined changes in allocation weights due to asset performance. We briefly considered whether such changes warranted rebalancing and what impact rebalancing might have on longer term portfolio returns given the drag from taxes. At the end, we asked what performance expectations we should have had to analyze results in the first place. It’s all well and good to say that a portfolio performed by “X”, but shouldn’t we have had a hypothesis as to how we expected the portfolio to perform?

Indeed we did. As we noted in the post, the historical averages method assumes the future will resemble of the past. But that is a rather nebulous statement. It begs the question, by how much? In this post, we aim to develop a framework to quantify the likelihood our portfolio will meet our expected performance targets. Among the many things discussed, the one that struck us most is the role simulation plays vis a vis backtesing. True, simulation has it’s own issues, but at least you’re not relying on a single trial to formulate a hypothesis. Basing an investment strategy on the results of what happened historically is, in many ways, basing it on a single trial no matter how much one believes history repeats, rhymes, or rinses.

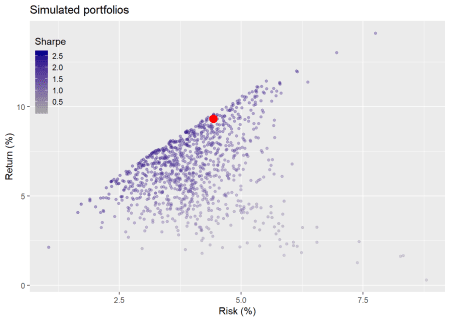
Enough tongue-wagging, let’s get to the data! Here’s our road map.

* Review the data
* Simulate possible return outcomes
* Incorporate portfolio weighting simulation
* Summarize and critique the simulations
* Next steps

**Review data**

Recall, our portfolio was comprised of the major asset classes: stocks, bonds, commodities (gold), and real estate. Data was pulled from the FRED database and began in 1987, the earliest date for a complete data set.

We took the average return and risk of those assets for the period of 1987 to 1991 and constructed 1,000 portfolios based on random weights. We then found all the portfolios that met our criteria of not less than a 7% return and not more than 10% risk on an annual basis. The average weights of those selected portfolios became our portfolio allocation for the test period. We calculated the return and risk for our portfolio (without rebalancing) during the first test period and compared that within another simulation of 1,000 possible portfolios. We graph that comparison below.



**First simulation**

If you’re thinking, why are we talking about more simulations when we’ve already done two rounds of them, it’s a legitimate question. The answer: we didn’t simulate potential returns and risk. Remember we took the historical five-year average return and risk of the assets and plugged those numbers into our weighting algorithm to simulate potential portfolios. We didn’t simulate the range of potential returns. It should be obvious that the returns we hoped to achieve based on our portfolio allocation were not likely to be the returns we would achieve. But what should we expect? Only 27% of the portfolios in our simulation would have achieved our risk-return constraints based on using the historical average as a proxy for our return expectation.

Is that an accurate probability? Hard to tell when we’re basing those return expectations on only one period. A more scientific way of looking at it could be to run a bunch of simulations to quantify how likely the risk and return constraints we chose would occur.

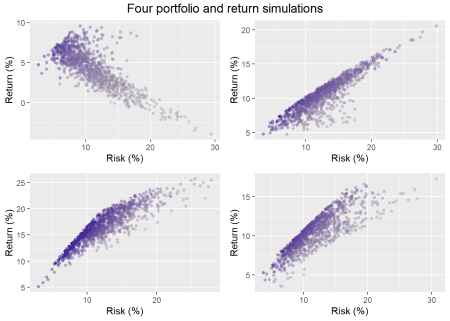
We’ll simulate 1,000 different return profiles of the four asset classes using their historical average return and risk. But instead of only using data from 1987, the earliest date in which we have a full data set, we’ll calculate the average based on as much of the data as is available. That does create another bias, but in this case only relates to real estate, which has the shortest period from our data source.

Before we proceed, we want to flag a problem with simulations: your assumptions around risk and return will be reproduced closely in the simulation without making an adjustment. Say you assume asset A has a 5% return and 12% risk. Guess what? Your simulation will have close to a 5% return and 12% risk. Thus we include a noise factor equivalent to a mean of zero and the asset’s standard deviation to account for uncertainty around our estimates.[1](https://osm.netlify.com/post/testing-expectations/#fn1)

Let’s sample four of the return simulations, run the portfolio allocation simulation on them, and graph the output.

Three of the simulations result in upwardly sloping risk-return portfolios, one does not. Finance theory posits that one should generally be compensated for taking more risk with higher returns. But there are cases where that doesn’t occur. And this isn’t just an artifact of randomness; it happens in reality too. Think of buying stocks before the tech or housing bubbles burst.

Looking in more detail at the graphs, one can see that even if the range of risk is close to the same, returns are not. Hence, the portfolio weights to achieve our risk-return constraints could be quite different, perhaps not even attainable. For example, in three out of four of our samples, just about any allocation would get you returns greater than 7% most of the time. But less than a third of the portfolios would meet the risk constraint. Recall our previous simulation suggested our probability of meeting our risk-return constraints was 27%. Indeed, based on these four samples the likelihood of meeting our risk-return constraints are 8%, 17%, 25%, and 26%. Not a great result.



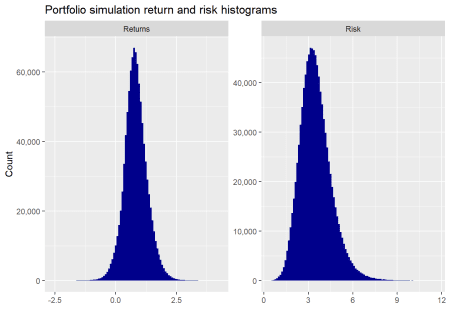
**Second simulation**

Critically, these results were only four randomly selected samples from 1,000 simulations. We want to run the portfolio allocation simulation on each of the return simulations. That would yield about a million different portfolios.

Let’s run these simulations and then graph the histograms. We won’t graph the risk-return scatter plot as we did above for the entire simulation because the results look like a snowstorm in Alaska. Instead, we’ll sample 10,000 portfolios and graph them in the usual way.



A wide scattering of returns and risk, as one might expect.Here are the return and risk histograms of the million portfolios.



Portfolio returns are close to normally distributed (though that is mostly due to the simulation), while risk is skewed due to the way portfolio volatility is calculated. Suffice it to say, these distributions aren’t unusual. But we should be particularly mindful of the return distribution, which we’ll discuss later.

The average return and risk are 10% and 12% on an annualized basis. That risk level suggests a lower likelihood of hitting our risk- return constraints. Indeed. when we run the calculation, only 19% of the portfolios achieve our goal, lower than our first simulation.

**Takeaways**

This test offers some solid insights into our original portfolio allocation. First, while our chosen constraints were only achieved in 27% of the portfolios. That didn’t tell us how likely those constraints were to occur over many possible outcomes. Even after the first five-year test, there wasn’t evidence that our constraints were unrealistic because of the time period in which the test fell: namely, the early-to-mid part of the tech boom.

When we ran the second five-year test our portfolio missed our targets. But could we say with confidence that such an outcome was a fluke? It wasn’t until we ran the return and portfolio allocation simulations together that we were able to quantify the likelihood of our targets. As noted, only 19% were likely to meet our constraints. Armed with that knowledge, we should revise our constraints either by lowering our return expectation or increasing our risk tolerance.

If we had run that simulation before our first test, it would have been relatively obvious that the performance was unusual. Indeed, the greater than 9% return with low risk that the portfolio achieved occurred only 14% of the time in our portfolio simulations. The upshot: not impossible, but not all that likely either.

From these results it seems clear that one should be cautious about the period used for a backtest. Clearly, we want to see how the portfolio would have performed historically. But that is only one result from the lab, so to speak.

**Critique**

While the test results are useful, there are a couple of problems with the analysis we want to point out. First, recall that our portfolio weightings assume we invest some amount in all available assets. That could increase the probability of meeting our constraints. But we would also want to be careful about how much we should trust that increased probabilty: it’s likely we could get at least a five percentage point improvement simply due to greater randomness.

Second, while we simulated asset returns, we did not simulate correlations,. Low correlation among assets reduces portfolio volatility, though variance usually has a bigger impact. But asset correlations are not stable over time. Hence, for completeness our simulations should include the impact of random changes in correlation. We would hope that the random draws implicitly incorporate randomness in correlations, but we’d need to run some calculations to confirm. Alternatively, we could explicitly attempt to randomize the correlations, but that is not as easy as it sounds and can lead to counter-intuitive, or even next to impossible results.[3](https://osm.netlify.com/post/testing-expectations/#fn3)

Third, our simulations drew from a normal distribution and most folks know that returns distributions for financial assets are not normal. One way to solve this is to sample from historical returns. If the sample length and timing were the same across the assets, then we would have partly addressed the correlation issue too. But, as noted before, our data series are not equal in length. Hence, we’d be introducing some bias if we only sampled from those periods in which we had data for every asset.

Fourth, historical returns are not independent of one another. In other words, today’s return on stock ABC, relates, to varying degrees, on ABC’s returns from yesterday or even further out, known as serial correlation. Similarly, ABC’s returns today exhibit some relation to yesterday’s returns for stock DEF, bond, GHI, and so on, known as cross-correlation.

Even if we’re able to find a sufficiently long and clean data set, we need to recognize that the returns in that sample, are still that, a sample, and thus not representative of of the full range of possible outcomes (whatever that applies in your cosmology). One need only compare historical returns prior to and after the onset of the covid-19 pandemic to attest to the fact that unexpected return patterns still occur more often than expected in financial data. There a bunch of ways to deal with this including more sophisticated sampling methods, imputing distributions using Bayesian approaches, We’ll cover the last one in detail in our next post! Kidding aside, we’re a big fan of Prof. Hyndman’s work. But the point is handling unexpected returns requires more and more sophistication. Addressing the merits of adding such sophistication is best left for another post.

**Next steps**

Let’s review. Over the past seven posts we looked at different methods to set return expectations for assets we might want to include in our portfolio. We then used the simplest method—historical averages—as the foundation for our asset allocation. From there, we simulated a range of possible portfolios, and chose an asset weighting to create a portfolio that would meet our risk-return criteria. We then allocated the assets accordingly and looked at how it performed over two five-year test periods. We noted that rebalancing or exclusion of assets might be warranted, but shelved those temporarily. We then tested how reasonable our risk-return expectations were and found that, in general, they seemed a bit unrealistic. What we haven’t looked at is whether choosing portfolio weights ends up producing a better outcome than the naive equal-weighted portfolio either historically or in simulation. And we also haven’t looked at finding an optimal portfolio for a given level of risk or return. Stay tuned.

Here’s the R followed by the Python code used to produce this post. Order does not reflect preference.

R code:

# Built using R 3.6.2

## Load packages

suppressPackageStartupMessages({

library(tidyquant)

library(tidyverse)

library(grid)

})

## Load data

symbols <- c("WILL5000INDFC", "BAMLCC0A0CMTRIV", "GOLDPMGBD228NLBM", "CSUSHPINSA", "DGS5")

sym\_names <- c("stock", "bond", "gold", "realt", "rfr")

getSymbols(symbols, src="FRED", from = "1970-01-01", to = "2019-12-31")

for(i in 1:5){

x <- getSymbols(symbols[i], src = "FRED",

from = "1970-01-01", to = "2019-12-31",

auto.assign = FALSE)

x <- to.monthly(x, indexAt ="lastof", OHLC = FALSE)

assign(sym\_names[i],x)

}

dat <- merge(stock, bond, gold, realt, rfr)

colnames(dat) <- sym\_names

dat <- dat["1970/2019"]

## create data frame

df <- data.frame(date = index(dat), coredata(dat)) %>%

mutate\_at(vars(-c(date, rfr)), function(x) x/lag(x)-1) %>%

mutate(rfr = rfr/100)

df <- df %>% filter(date>="1987-01-01")

## Load simuation function

port\_sim <- function(df, sims, cols){

if(ncol(df) != cols){

print("Columns don't match")

break

}

# Create weight matrix

wts <- matrix(nrow = sims, ncol = cols)

for(i in 1:sims){

a <- runif(cols,0,1)

b <- a/sum(a)

wts[i,] <- b

}

# Find returns

mean\_ret <- colMeans(df)

# Calculate covariance matrix

cov\_mat <- cov(df)

# Calculate random portfolios

port <- matrix(nrow = sims, ncol = 2)

for(i in 1:sims){

port[i,1] <- as.numeric(sum(wts[i,] \* mean\_ret))

port[i,2] <- as.numeric(sqrt(t(wts[i,]) %\*% cov\_mat %\*% wts[i,]))

}

colnames(port) <- c("returns", "risk")

port <- as.data.frame(port)

port$Sharpe <- port$returns/port$risk\*sqrt(12)

max\_sharpe <- port[which.max(port$Sharpe),]

graph <- port %>%

ggplot(aes(risk\*sqrt(12)\*100, returns\*1200, color = Sharpe)) +

geom\_point(size = 1.2, alpha = 0.4) +

scale\_color\_gradient(low = "darkgrey", high = "darkblue") +

labs(x = "Risk (%)",

y = "Return (%)",

title = "Simulated portfolios")

out <- list(port = port, graph = graph, max\_sharpe = max\_sharpe, wts = wts)

}

## Run simuation

set.seed(123)

port\_sim\_1 <- port\_sim(df\_old[2:61,2:5],1000,4)

## Load portfolio selection function

port\_select\_func <- function(port, return\_min, risk\_max, port\_names){

port\_select <- cbind(port$port, port$wts)

port\_wts <- port\_select %>%

mutate(returns = returns\*12,

risk = risk\*sqrt(12)) %>%

filter(returns >= return\_min,

risk <= risk\_max) %>%

summarise\_at(vars(4:7), mean) %>%

`colnames<-`(port\_names)

graph <- port\_wts %>%

rename("Stocks" = 1,

"Bonds" = 2,

"Gold" = 3,

"Real estate" = 4) %>%

gather(key,value) %>%

ggplot(aes(reorder(key,value), value\*100 )) +

geom\_bar(stat='identity', position = "dodge", fill = "blue") +

geom\_text(aes(label=round(value,2)\*100), vjust = -0.5) +

scale\_y\_continuous(limits = c(0,40)) +

labs(x="",

y = "Weights (%)",

title = "Average weights for risk-return constraints")

out <- list(port\_wts = port\_wts, graph = graph)

out

}

## Run selection function

results\_1 <- port\_select\_func(port\_sim\_1,0.07, 0.1, sym\_names[1:4])

## Calculate probability of success

port\_success <- round(mean(port\_sim\_1$port$returns > 0.07/12 &

port\_sim\_1$port$risk <= 0.1/sqrt(12)),3)\*100

## Function for portfolio returns without rebalancing

rebal\_func <- function(act\_ret, weights){

ret\_vec <- c()

wt\_mat <- matrix(nrow = nrow(act\_ret), ncol = ncol(act\_ret))

for(i in 1:60){

wt\_ret <- act\_ret[i,]\*weights # wt'd return

ret <- sum(wt\_ret) # total return

ret\_vec[i] <- ret

weights <- (weights + wt\_ret)/(sum(weights)+ret) # new weight based on change in asset value

wt\_mat[i,] <- as.numeric(weights)

}

out <- list(ret\_vec = ret\_vec, wt\_mat = wt\_mat)

out

}

## Run function and create actual portfolio and data frame for graph

port\_1\_act <- rebal\_func(df[62:121,2:5],results\_1$port\_wts)

port\_act <- data.frame(returns = mean(port\_1\_act$ret\_vec),

risk = sd(port\_1\_act$ret\_vec),

sharpe = mean(port\_1\_act$ret\_vec)/sd(port\_1\_act$ret\_vec)\*sqrt(12))

## Simulate portfolios on first five-year period

set.seed(123)

port\_sim\_2 <- port\_sim(df[62:121,2:5], 1000, 4)

## Graph simulation with chosen portfolio

port\_sim\_2$graph +

geom\_point(data = port\_act,

aes(risk\*sqrt(12)\*100, returns\*1200),

size = 4,

color="red") +

theme(legend.position = c(0.05,0.8), legend.key.size = unit(.5, "cm"),

legend.background = element\_rect(fill = NA))

## Load longer term data

symbols <- c("WILL5000INDFC", "BAMLCC0A0CMTRIV", "GOLDPMGBD228NLBM", "CSUSHPINSA", "DGS5")

sym\_names <- c("stock", "bond", "gold", "realt", "rfr")

for(i in 1:5){

x <- getSymbols(symbols[i], src = "FRED",

from = "1970-01-01", to = "2019-12-31",

auto.assign = FALSE)

x <- to.monthly(x, indexAt ="lastof", OHLC = FALSE)

assign(sym\_names[i],x)

}

dat <- merge(stock, bond, gold, realt, rfr)

colnames(dat) <- sym\_names

dat <- dat["1970/2019"]

## create data frame

df <- data.frame(date = index(dat), coredata(dat)) %>%

mutate\_at(vars(-c(date, rfr)), function(x) x/lag(x)-1) %>%

mutate(rfr = rfr/100)

df <- df %>% filter(date>="1971-01-01")

## Load long data

df <- readRDS('port\_const\_long.rds')

## Prepare sample

hist\_avg <- df %>%

filter(date <= "1991-12-31") %>%

summarise\_at(vars(-date), list(mean = function(x) mean(x, na.rm=TRUE),

sd = function(x) sd(x, na.rm = TRUE))) %>%

gather(key, value) %>%

mutate(key = str\_remove(key, "\_.\*"),

key = factor(key, levels =sym\_names)) %>%

mutate(calc = c(rep("mean",5), rep("sd",5))) %>%

spread(calc, value)

# Run simulation

set.seed(123)

sim1 <- list()

for(i in 1:1000){

a <- rnorm(60, hist\_avg[1,2], hist\_avg[1,3]) + rnorm(60, 0, hist\_avg[1,3])

b <- rnorm(60, hist\_avg[2,2], hist\_avg[2,3]) + rnorm(60, 0, hist\_avg[2,3])

c <- rnorm(60, hist\_avg[3,2], hist\_avg[3,3]) + rnorm(60, 0, hist\_avg[3,3])

d <- rnorm(60, hist\_avg[4,2], hist\_avg[4,3]) + rnorm(60, 0, hist\_avg[4,3])

df1 <- data.frame(a, b, c, d)

cov\_df1 <- cov(df1)

sim1[[i]] <- list(df1, cov\_df1)

names(sim1[[i]]) <- c("df", "cov\_df")

}

# Sample four return paths

# Note this is the code we used to create the graphs that worked repeatedly when we ran the source code, but would not reproduce accurately in blogdown. Don't know if anyone else has experienced this issue. But after multiple attempts including repasting the code and even seeing the same results when run within the Rmarkdown file, we gave up and read the saved example into blogdown. Let us know if you can't reproduce it.

set.seed(123)

grafs <- list()

for(i in 1:4){

samp <- sample(1000,1)

grafs[[i]] <- port\_sim(sim1[[samp]]$df,1000,4)

}

gridExtra::grid.arrange(grafs[[1]]$graph +

theme(legend.position = "none") +

labs(title = NULL),

grafs[[2]]$graph +

theme(legend.position = "none") +

labs(title = NULL),

grafs[[3]]$graph +

theme(legend.position = "none") +

labs(title = NULL),

grafs[[4]]$graph +

theme(legend.position = "none") +

labs(title = NULL),

ncol=2, nrow=2,

top = textGrob("Four portfolio and return simulations",gp=gpar(fontsize=12)))

# Calculate probability of hitting risk-return constraint

probs <- c()

for(i in 1:4){

probs[i] <- round(mean(grafs[[i]]$port$returns >= 0.07/12 &

grafs[[i]]$port$risk <=0.1/sqrt(12)),2)\*100

}

probs

## Run simulation

# Portfolio weight function

wt\_func <- function(){

a <- runif(4,0,1)

a/sum(a)

}

# Portfolio volatility calculation

vol\_calc <- function(cov\_dat, weights){

sqrt(t(weights) %\*% cov\_dat %\*% weights)

}

# Simulate

# Note this should run pretty quickly, at least less than a minute.

set.seed(123)

portfolios <- list()

for(i in 1:1000){

wt\_mat <- as.matrix(t(replicate(1000, wt\_func(), simplify = "matrix")))

avg\_ret <- colMeans(sim1[[i]]$df)

returns <- as.vector(avg\_ret %\*% t(wt\_mat))

cov\_dat <- cov(sim1[[i]]$df)

risk <- apply(wt\_mat, 1, function(x) vol\_calc(cov\_dat,x))

portfolios[[i]] <- data.frame(returns, risk)

}

port\_1m <- do.call("rbind", portfolios)

port\_1m\_prob <- round(mean(port\_1m$returns\*12 >= 0.07 & port\_1m$risk\*sqrt(12) <= 0.1),2)\*100

## Graph sample of port\_1m

set.seed(123)

port\_samp = port\_1m[sample(1e6, 1e4),] # choose 10,000 samples from 1,000,000 portfolios.

port\_samp %>%

mutate(Sharpe = returns/risk) %>%

ggplot(aes(risk\*sqrt(12)\*100, returns\*1200, color = Sharpe)) +

geom\_point(size = 1.2, alpha = 0.4) +

scale\_color\_gradient(low = "darkgrey", high = "darkblue") +

labs(x = "Risk (%)",

y = "Return (%)",

title = "Ten thousand samples from simulation of one million portfolios") +

theme(legend.position = c(0.05,0.8), legend.key.size = unit(.5, "cm"),

legend.background = element\_rect(fill = NA))

# Graph histograms

port\_1m %>%

mutate(returns = returns\*100,

risk = risk\*100) %>%

gather(key, value) %>%

ggplot(aes(value)) +

geom\_histogram(bins = 100, fill = "darkblue") +

facet\_wrap(~key, scales = "free",

labeller = as\_labeller(c(returns = "Returns",

risk = "Risk"))) +

scale\_y\_continuous(labels = scales::comma) +

labs(x = "",

y = "Count",

title = "Portfolio simulation return and risk histograms")

Python code:

# Built using Python 3.7.4

# Load libraries

import pandas as pd

import pandas\_datareader.data as web

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

plt.style.use('ggplot')

sns.set()

# Load data

start\_date = '1970-01-01'

end\_date = '2019-12-31'

symbols = ["WILL5000INDFC", "BAMLCC0A0CMTRIV", "GOLDPMGBD228NLBM", "CSUSHPINSA", "DGS5"]

sym\_names = ["stock", "bond", "gold", "realt", 'rfr']

filename = 'data\_port\_const.pkl'

try:

df = pd.read\_pickle(filename)

print('Data loaded')

except FileNotFoundError:

print("File not found")

print("Loading data", 30\*"-")

data = web.DataReader(symbols, 'fred', start\_date, end\_date)

data.columns = sym\_names

data\_mon = data.resample('M').last()

df = data\_mon.pct\_change()['1987':'2019']

df.to\_pickle(filename) # If you haven't saved the file

df = data\_mon.pct\_change()['1971':'2019']

pd.to\_pickle(df,filename) # if you haven't saved the file

## Simulation function

class Port\_sim:

def calc\_sim(df, sims, cols):

wts = np.zeros((sims, cols))

for i in range(sims):

a = np.random.uniform(0,1,cols)

b = a/np.sum(a)

wts[i,] = b

mean\_ret = df.mean()

port\_cov = df.cov()

port = np.zeros((sims, 2))

for i in range(sims):

port[i,0] = np.sum(wts[i,]\*mean\_ret)

port[i,1] = np.sqrt(np.dot(np.dot(wts[i,].T,port\_cov), wts[i,]))

sharpe = port[:,0]/port[:,1]\*np.sqrt(12)

best\_port = port[np.where(sharpe == max(sharpe))]

max\_sharpe = max(sharpe)

return port, wts, best\_port, sharpe, max\_sharpe

def calc\_sim\_lv(df, sims, cols):

wts = np.zeros(((cols-1)\*sims, cols))

count=0

for i in range(1,cols):

for j in range(sims):

a = np.random.uniform(0,1,(cols-i+1))

b = a/np.sum(a)

c = np.random.choice(np.concatenate((b, np.zeros(i))),cols, replace=False)

wts[count,] = c

count+=1

mean\_ret = df.mean()

port\_cov = df.cov()

port = np.zeros(((cols-1)\*sims, 2))

for i in range(sims):

port[i,0] = np.sum(wts[i,]\*mean\_ret)

port[i,1] = np.sqrt(np.dot(np.dot(wts[i,].T,port\_cov), wts[i,]))

sharpe = port[:,0]/port[:,1]\*np.sqrt(12)

best\_port = port[np.where(sharpe == max(sharpe))]

max\_sharpe = max(sharpe)

return port, wts, best\_port, sharpe, max\_sharpe

def graph\_sim(port, sharpe):

plt.figure(figsize=(14,6))

plt.scatter(port[:,1]\*np.sqrt(12)\*100, port[:,0]\*1200, marker='.', c=sharpe, cmap='Blues')

plt.colorbar(label='Sharpe ratio', orientation = 'vertical', shrink = 0.25)

plt.title('Simulated portfolios', fontsize=20)

plt.xlabel('Risk (%)')

plt.ylabel('Return (%)')

plt.show()

# Plot

np.random.seed(123)

port\_sim\_1, wts\_1, \_, sharpe\_1, \_ = Port\_sim.calc\_sim(df.iloc[1:60,0:4],1000,4)

Port\_sim.graph\_sim(port\_sim\_1, sharpe\_1)

## Selection function

# Constraint function

def port\_select\_func(port, wts, return\_min, risk\_max):

port\_select = pd.DataFrame(np.concatenate((port, wts), axis=1))

port\_select.columns = ['returns', 'risk', 1, 2, 3, 4]

port\_wts = port\_select[(port\_select['returns']\*12 >= return\_min) & (port\_select['risk']\*np.sqrt(12) <= risk\_max)]

port\_wts = port\_wts.iloc[:,2:6]

port\_wts = port\_wts.mean(axis=0)

def graph():

plt.figure(figsize=(12,6))

key\_names = {1:"Stocks", 2:"Bonds", 3:"Gold", 4:"Real estate"}

lab\_names = []

graf\_wts = port\_wts.sort\_values()\*100

for i in range(len(graf\_wts)):

name = key\_names[graf\_wts.index[i]]

lab\_names.append(name)

plt.bar(lab\_names, graf\_wts)

plt.ylabel("Weight (%)")

plt.title("Average weights for risk-return constraint", fontsize=15)

for i in range(len(graf\_wts)):

plt.annotate(str(round(graf\_wts.values[i])), xy=(lab\_names[i], graf\_wts.values[i]+0.5))

plt.show()

return port\_wts, graph()

# Graph

results\_1\_wts,\_ = port\_select\_func(port\_sim\_1, wts\_1, 0.07, 0.1)

# Return function with no rebalancing

def rebal\_func(act\_ret, weights):

ret\_vec = np.zeros(len(act\_ret))

wt\_mat = np.zeros((len(act\_ret), len(act\_ret.columns)))

for i in range(len(act\_ret)):

wt\_ret = act\_ret.iloc[i,:].values\*weights

ret = np.sum(wt\_ret)

ret\_vec[i] = ret

weights = (weights + wt\_ret)/(np.sum(weights) + ret)

wt\_mat[i,] = weights

return ret\_vec, wt\_mat

## Run rebalance function using desired weights

port\_1\_act, wt\_mat = rebal\_func(df.iloc[61:121,0:4], results\_1\_wts)

port\_act = {'returns': np.mean(port\_1\_act),

'risk': np.std(port\_1\_act),

'sharpe': np.mean(port\_1\_act)/np.std(port\_1\_act)\*np.sqrt(12)}

# Run simulation on first five-year period

np.random.seed(123)

port\_sim\_2, wts\_2, \_, sharpe\_2, \_ = Port\_sim.calc\_sim(df.iloc[61:121,0:4],1000,4)

# Graph simulation with actual portfolio return

plt.figure(figsize=(14,6))

plt.scatter(port\_sim\_2[:,1]\*np.sqrt(12)\*100, port\_sim\_2[:,0]\*1200, marker='.', c=sharpe\_2, cmap='Blues')

plt.colorbar(label='Sharpe ratio', orientation = 'vertical', shrink = 0.25)

plt.scatter(port\_act['risk']\*np.sqrt(12)\*100, port\_act['returns']\*1200, c='red', s=50)

plt.title('Simulated portfolios', fontsize=20)

plt.xlabel('Risk (%)')

plt.ylabel('Return (%)')

plt.show()

# Calculate returns and risk for longer period

# Call prior file for longer period

df = data\_mon.pct\_change()['1971':'2019']

hist\_mu = df['1971':'1991'].mean(axis=0)

hist\_sigma = df['1971':'1991'].std(axis=0)

# Run simulation based on historical figures

np.random.seed(123)

sim1 = []

for i in range(1000):

#np.random.normal(mu, sigma, obs)

a = np.random.normal(hist\_mu[0], hist\_sigma[0], 60) + np.random.normal(0, hist\_sigma[0], 60)

b = np.random.normal(hist\_mu[1], hist\_sigma[1], 60) + np.random.normal(0, hist\_sigma[1], 60)

c = np.random.normal(hist\_mu[2], hist\_sigma[2], 60) + np.random.normal(0, hist\_sigma[2], 60)

d = np.random.normal(hist\_mu[3], hist\_sigma[3], 60) + np.random.normal(0, hist\_sigma[3], 60)

df1 = pd.DataFrame(np.array([a, b, c, d]).T)

cov\_df1 = df1.cov()

sim1.append([df1, cov\_df1])

# create graph objects

np.random.seed(123)

graphs = []

for i in range(4):

samp = np.random.randint(1,1000)

port, \_, \_, sharpe, \_ = Port\_sim.calc\_sim(sim1[samp][0], 1000, 4)

graf = [port,sharpe]

graphs.append(graf)

# Graph sample portfolios

fig, axes = plt.subplots(2, 2, figsize=(12,6))

for i, ax in enumerate(fig.axes):

ax.scatter(graphs[i][0][:,1]\*np.sqrt(12)\*100, graphs[i][0][:,0]\*1200, marker='.', c=graphs[i][1], cmap='Blues')

plt.show()

# Calculate probability of hitting risk-return constraints based on sample portfolos

probs = []

for i in range(4):

out = round(np.mean((graphs[i][0][:,0] >= 0.07/12) & (graphs[i][0][:,1] <= 0.1/np.sqrt(12))),2)\*100

probs.append(out)

# Simulate portfolios from reteurn simulations

def wt\_func():

a = np.random.uniform(0,1,4)

return a/np.sum(a)

# Note this should run relatively quickly: less than a minute.

np.random.seed(123)

portfolios = np.zeros((1000, 1000, 2))

for i in range(1000):

wt\_mat = np.array([wt\_func() for \_ in range(1000)])

port\_ret = sim1[i][0].mean(axis=0)

cov\_dat = sim1[i][0].cov()

returns = np.dot(wt\_mat, port\_ret)

risk = [np.sqrt(np.dot(np.dot(wt.T,cov\_dat), wt)) for wt in wt\_mat]

portfolios[i][:,0] = returns

portfolios[i][:,1] = risk

port\_1m = portfolios.reshape((1000000,2))

# Find probability of hitting risk-return constraints on simulated portfolios

port\_1m\_prob = round(np.mean((port\_1m[:][:,0] > 0.07/12) & (port\_1m[:][:,1] <= 0.1/np.sqrt(12))),2)\*100

print(f"The probability of meeting our portfolio constraints is:{port\_1m\_prob: 0.0f}%")

# Plot sample portfolios

np.random.seed(123)

port\_samp = port\_1m[np.random.choice(1000000, 10000),:]

sharpe = port\_samp[:,0]/port\_samp[:,1]

plt.figure(figsize=(14,6))

plt.scatter(port\_samp[:,1]\*np.sqrt(12)\*100, port\_samp[:,0]\*1200, marker='.', c=sharpe, cmap='Blues')

plt.colorbar(label='Sharpe ratio', orientation = 'vertical', shrink = 0.25)

plt.title('Ten thousand samples from one million simulated portfolios', fontsize=20)

plt.xlabel('Risk (%)')

plt.ylabel('Return (%)')

plt.show()

# Graph histograms

fig, axes = plt.subplots(1,2, figsize = (12,6))

for idx,ax in enumerate(fig.axes):

if idx == 1:

ax.hist(port\_1m[:][:,1], bins = 100)

else:

ax.hist(port\_1m[:][:,0], bins = 100)

plt.show()