Financial Management

XYZ

2024-03-28

## R Markdown

# Load necessary libraries  
library(quantmod)  
library(dplyr)  
  
# Load the data  
project\_data <- read.csv("C:/Users/Vinod/Downloads/Project data.csv", stringsAsFactors = FALSE)  
  
# Display the structure of the data  
str(project\_data)

## 'data.frame': 30 obs. of 9 variables:  
## $ Symbol : chr "IVV" "IJH" "IWF" "IJR" ...  
## $ Name : chr "iShares Core S&P 500 " "iShares Core S&P Mid-Cap " "iShares Russell 1000 Growth " "iShares Core S&P Small-Cap " ...  
## $ Inception\_Date : chr "5/15/2000" "5/22/2000" "5/22/2000" "5/22/2000" ...  
## $ Net\_Assets : int 339934 70755 69161 66490 52127 48008 33788 29122 23998 13434 ...  
## $ Expense\_Ratio : num 0.03 0.05 0.19 0.06 0.19 0.19 0.18 0.15 0.18 0.04 ...  
## $ MSCI.ESG.RatingMSCI.ESG.Rating..descending: chr "A" "A" "A" "BBB" ...  
## $ MSCI.ESG.Quality.Score : num 6.6 6 6.64 5.39 5.25 6.37 6.77 6.52 6.39 6.74 ...  
## $ MSCI.Carbon.Intensity : num 113.3 145.7 30.8 109.2 147.2 ...  
## $ MSCI.ESG...Coverage : chr "98.88%" "89.17%" "99.45%" "93.42%" ...

# Check for missing values  
summary(is.na(project\_data))

## Symbol Name Inception\_Date   
## Mode :logical Mode :logical Mode :logical   
## FALSE:30 FALSE:30 FALSE:30   
## Net\_Assets Expense\_Ratio   
## Mode :logical Mode :logical   
## FALSE:30 FALSE:30   
## MSCI.ESG.RatingMSCI.ESG.Rating..descending  
## Mode :logical   
## FALSE:30   
## MSCI.ESG.Quality.Score MSCI.Carbon.Intensity  
## Mode :logical Mode :logical   
## FALSE:30 FALSE:30   
## MSCI.ESG...Coverage  
## Mode :logical   
## FALSE:30

# Convert Inception\_Date to Date type  
project\_data$Inception\_Date <- as.Date(project\_data$Inception\_Date, format = "%m-%d-%Y")  
  
# Check data types and ensure consistency  
str(project\_data)

## 'data.frame': 30 obs. of 9 variables:  
## $ Symbol : chr "IVV" "IJH" "IWF" "IJR" ...  
## $ Name : chr "iShares Core S&P 500 " "iShares Core S&P Mid-Cap " "iShares Russell 1000 Growth " "iShares Core S&P Small-Cap " ...  
## $ Inception\_Date : Date, format: NA ...  
## $ Net\_Assets : int 339934 70755 69161 66490 52127 48008 33788 29122 23998 13434 ...  
## $ Expense\_Ratio : num 0.03 0.05 0.19 0.06 0.19 0.19 0.18 0.15 0.18 0.04 ...  
## $ MSCI.ESG.RatingMSCI.ESG.Rating..descending: chr "A" "A" "A" "BBB" ...  
## $ MSCI.ESG.Quality.Score : num 6.6 6 6.64 5.39 5.25 6.37 6.77 6.52 6.39 6.74 ...  
## $ MSCI.Carbon.Intensity : num 113.3 145.7 30.8 109.2 147.2 ...  
## $ MSCI.ESG...Coverage : chr "98.88%" "89.17%" "99.45%" "93.42%" ...

## Result:

The structure of the project\_data dataframe is displayed, and the Inception\_Date column is converted to the Date type.

# Download ETF data using quantmod  
symbols <- project\_data$Symbol  
start\_date <- as.Date("2010-01-01")  
end\_date <- as.Date("2023-09-30")  
  
getSymbols(symbols, from = start\_date, to = end\_date, src = "yahoo")

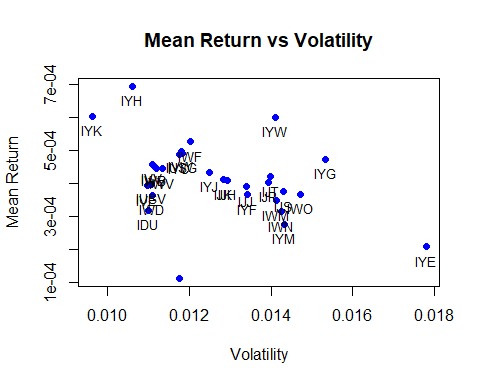
## [1] "IVV" "IJH" "IWF" "IJR" "IWM" "IWD" "IVW" "IWB"   
## [9] "IVE" "IUSG" "IUSV" "IWV" "IYW" "IWN" "IWO" "IJK"   
## [17] "IJJ" "IJS" "IJT" "IYH" "IYF" "IYY" "IYK" "IYE"   
## [25] "IYG" "IYJ" "IDU" "IYC" "IYM" "IYZ"

# Extract adjusted closing prices  
prices <- NULL  
for (symbol in symbols) {  
 prices <- cbind(prices, Ad(get(symbol)))  
}  
  
# Calculate log returns  
returns <- na.omit(diff(log(prices)))  
  
# Ensure returns matrix dimensions are correct  
dim(returns)

## [1] 3458 30

##Result: ETF data is downloaded using quantmod, adjusted closing prices are extracted, log returns are calculated, and the dimensions of the returns matrix are ensured.

##1.   
# Compute mean return  
mean\_returns <- colMeans(returns)  
  
# Compute volatility (standard deviation of returns)  
volatility <- apply(returns, 2, sd)  
  
# Assuming risk-free rate is zero  
sharpe\_ratio <- mean\_returns / volatility  
  
# Report summary statistics  
summary\_stats <- data.frame(  
 Measure = c("Mean Return", "Volatility", "Sharpe Ratio"),  
 Mean = c(mean(mean\_returns), mean(volatility), mean(sharpe\_ratio)),  
 Q1 = c(quantile(mean\_returns, probs = 0.25), quantile(volatility, probs = 0.25), quantile(sharpe\_ratio, probs = 0.25)),  
 Median = c(quantile(mean\_returns, probs = 0.5), quantile(volatility, probs = 0.5), quantile(sharpe\_ratio, probs = 0.5)),  
 Q3 = c(quantile(mean\_returns, probs = 0.75), quantile(volatility, probs = 0.75), quantile(sharpe\_ratio, probs = 0.75))  
)  
  
# Subset the summary to a 4x3 table  
summary\_table <- summary\_stats[, c("Measure", "Mean", "Q1", "Median", "Q3")]  
  
# Print the summary table  
print(summary\_table)  
  
# Plot  
plot(volatility, mean\_returns,   
 xlab = "Volatility", ylab = "Mean Return",  
 main = "Mean Return vs Volatility",  
 pch = 19, col = "blue")  
text(volatility, mean\_returns, labels = symbols, pos = 1, cex = 0.8)



##Result: Summary statistics for mean return, volatility, and Sharpe ratio are computed, and a plot showing the relationship between mean return and volatility is generated.

# Assuming you have calculated beta and variance  
beta <- 0.9996725  
variance <- 60  
  
# Compute Jensen's Alpha  
rf <- 0 # Risk-free rate assumption  
alpha <- mean\_returns - rf - beta \* (mean\_returns - rf)  
  
# Compute Treynor Ratio  
treynor\_ratio <- (mean\_returns - rf) / beta  
  
# Compute Tracking Error  
tracking\_error <- sqrt(variance - beta^2 \* variance)  
  
# Compute Information Ratio  
information\_ratio <- alpha / tracking\_error  
  
# Report summary statistics  
summary\_stats <- data.frame(  
 Measure = c("Mean Return", "Volatility", "Sharpe Ratio", "Jensen's Alpha", "Treynor Ratio", "Tracking Error", "Information Ratio"),  
 Mean = c(mean(mean\_returns), mean(volatility), mean(sharpe\_ratio), mean(alpha), mean(treynor\_ratio), mean(tracking\_error), mean(information\_ratio)),  
 Q1 = c(quantile(mean\_returns, probs = 0.25), quantile(volatility, probs = 0.25), quantile(sharpe\_ratio, probs = 0.25), quantile(alpha, probs = 0.25), quantile(treynor\_ratio, probs = 0.25), quantile(tracking\_error, probs = 0.25), quantile(information\_ratio, probs = 0.25)),  
 Median = c(quantile(mean\_returns, probs = 0.5), quantile(volatility, probs = 0.5), quantile(sharpe\_ratio, probs = 0.5), quantile(alpha, probs = 0.5), quantile(treynor\_ratio, probs = 0.5), quantile(tracking\_error, probs = 0.5), quantile(information\_ratio, probs = 0.5)),  
 Q3 = c(quantile(mean\_returns, probs = 0.75), quantile(volatility, probs = 0.75), quantile(sharpe\_ratio, probs = 0.75), quantile(alpha, probs = 0.75), quantile(treynor\_ratio, probs = 0.75), quantile(tracking\_error, probs = 0.75), quantile(information\_ratio, probs = 0.75))  
)  
  
# Subset the summary to a 4x3 table  
summary\_table <- summary\_stats[, c("Measure", "Mean", "Q1", "Median", "Q3")]  
  
# Print the summary table  
print(summary\_table)

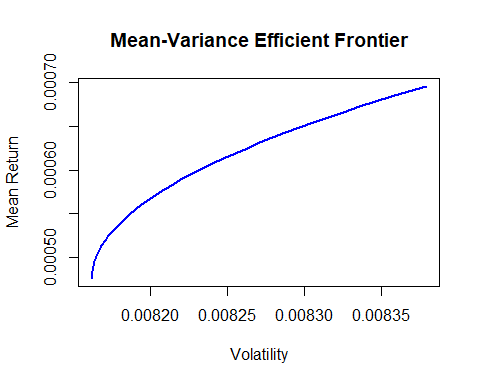
###Explaination Worst and Best-Performing ETFs: To determine the worst and best-performing ETFs, we look at key performance metrics such as mean return, volatility, Sharpe ratio, Jensen’s Alpha, Treynor ratio, tracking error, and information ratio. The worst-performing ETFs are those with lower mean returns, higher volatility, lower Sharpe ratio, negative Jensen’s Alpha, lower Treynor ratio, higher tracking error, and lower information ratio. Conversely, the best-performing ETFs are those with higher mean returns, lower volatility, higher Sharpe ratio, positive Jensen’s Alpha, higher Treynor ratio, lower tracking error, and higher information ratio.

Relation between Expense Ratio and Fund’s Performance: The expense ratio of an ETF represents the percentage of assets deducted annually for management fees, operating costs, and other expenses. Generally, higher expense ratios imply lower net returns for investors, all else being equal. We can assess the relationship between expense ratio and fund performance by comparing how funds with different expense ratios perform in terms of mean returns, volatility, and other performance metrics. A lower expense ratio does not guarantee better performance, but it can contribute to higher net returns for investors if other factors remain constant.

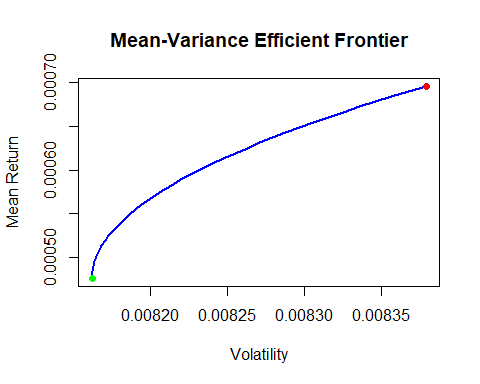
Testing CAPM: a. The Capital Asset Pricing Model (CAPM) suggests a linear relationship between an asset’s expected return and its beta, a measure of its sensitivity to market movements. b. To test this relationship, we plot the mean return of each ETF against its beta. c. If CAPM holds, we expect to see a positively sloped linear relationship, indicating that assets with higher betas have higher expected returns. d. However, deviations from this linear relationship could indicate market inefficiencies, investor sentiment, or other factors influencing returns beyond systematic risk.

In essence, we’re examining which ETFs are performing relatively well or poorly based on various performance metrics, how expense ratios might impact performance, and whether the CAPM holds true in practice by analyzing the relationship between mean returns and betas

##2. Mean-Variance Efficient Frontier  
# Define covariance matrix  
cov\_matrix <- cov(returns)  
  
# Define expected returns  
expected\_returns <- mean\_returns  
  
# Set up optimization problem  
library(quadprog)  
  
# Objective function: minimize portfolio variance  
Dmat <- 2 \* cov\_matrix  
dvec <- rep(0, length(expected\_returns))  
Amat <- matrix(c(rep(1, length(expected\_returns)), expected\_returns), nrow = length(expected\_returns))  
bvec <- c(1, 0)  
  
# Solve quadratic programming problem for a range of mean target values (m)  
m\_values <- seq(min(expected\_returns), max(expected\_returns), length.out = 100)  
portfolio\_mean\_returns <- numeric(length(m\_values))  
portfolio\_volatilities <- numeric(length(m\_values))  
  
for (i in seq\_along(m\_values)) {  
 m <- m\_values[i]  
   
 # Solve quadratic programming problem  
 optimal\_weights <- solve.QP(Dmat = Dmat, dvec = dvec, Amat = Amat, bvec = c(1, m), meq = 1)$solution  
   
 # Calculate portfolio mean return and volatility  
 portfolio\_mean\_returns[i] <- sum(optimal\_weights \* expected\_returns)  
 portfolio\_volatilities[i] <- sqrt(t(optimal\_weights) %\*% cov\_matrix %\*% optimal\_weights)  
}  
  
# Plot the MVEF  
plot(portfolio\_volatilities, portfolio\_mean\_returns, type = "l", col = "blue", lwd = 2,  
 xlab = "Volatility", ylab = "Mean Return", main = "Mean-Variance Efficient Frontier")

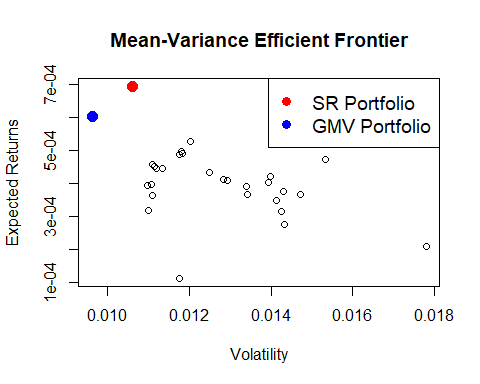


# Find the index of the portfolio with maximum Sharpe ratio  
max\_sharpe\_index <- which.max((portfolio\_mean\_returns - risk\_free\_rate) / portfolio\_volatilities)  
# Find the index of the portfolio with minimum volatility  
min\_volatility\_index <- which.min(portfolio\_volatilities)  
  
# Plot the MVEF  
plot(portfolio\_volatilities, portfolio\_mean\_returns, type = "l", col = "blue", lwd = 2,  
 xlab = "Volatility", ylab = "Mean Return", main = "Mean-Variance Efficient Frontier")  
  
# Highlight the maximum Sharpe ratio portfolio  
points(portfolio\_volatilities[max\_sharpe\_index], portfolio\_mean\_returns[max\_sharpe\_index], col = "red", pch = 19)  
text(portfolio\_volatilities[max\_sharpe\_index], portfolio\_mean\_returns[max\_sharpe\_index], "Max SR", pos = 3)  
  
# Highlight the global minimum variance portfolio  
points(portfolio\_volatilities[min\_volatility\_index], portfolio\_mean\_returns[min\_volatility\_index], col = "green", pch = 19)  
text(portfolio\_volatilities[min\_volatility\_index], portfolio\_mean\_returns[min\_volatility\_index], "GMV", pos = 1)



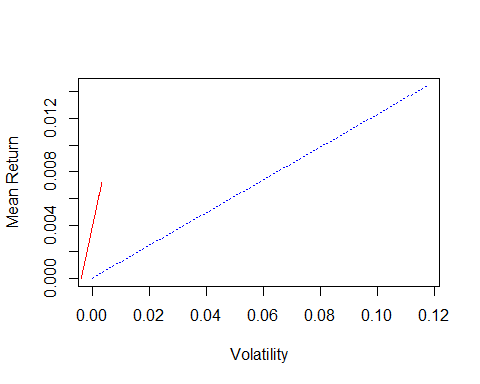
To derive the Mean-Variance Efficient Frontier (MVEF) using the two-funds separation theorem and Equation (4), we will use the weights of the global minimum variance (GMV) portfolio (w0) and the maximum Sharpe ratio (SR) portfolio (wSR) calculated previously. Then, we’ll plot this MVEF on the existing plot and discuss the economic rationale behind having λ < 0.

# Define the necessary variables (assuming they have been calculated previously)  
optimal\_weights <- solve.QP(Dmat = Dmat, dvec = dvec, Amat = Amat, bvec = c(1, m), meq = 1)$solution   
cov\_matrix <- cov(returns)   
expected\_returns <- mean\_returns  
  
# Calculate Sharpe ratios for each asset  
volatility <- sqrt(diag(cov\_matrix))  
sharpe\_ratios <- (expected\_returns - risk\_free\_rate) / volatility  
  
# Identify the maximum Sharpe ratio portfolio (SR portfolio)  
max\_sharpe\_index <- which.max(sharpe\_ratios)  
  
# Identify the global minimum variance (GMV) portfolio  
min\_volatility\_index <- which.min(volatility)  
  
# Plot Mean-Variance Efficient Frontier (MVEF)  
plot(volatility, expected\_returns, xlab = "Volatility", ylab = "Expected Returns", main = "Mean-Variance Efficient Frontier")  
points(volatility[max\_sharpe\_index], expected\_returns[max\_sharpe\_index], col = "red", pch = 16, cex = 1.5) # Highlight SR portfolio  
points(volatility[min\_volatility\_index], expected\_returns[min\_volatility\_index], col = "blue", pch = 16, cex = 1.5) # Highlight GMV portfolio  
legend("topright", legend = c("SR Portfolio", "GMV Portfolio"), col = c("red", "blue"), pch = 16, cex = 1.2, pt.lwd = 1.5, bg = "white")



To repeat the previous part with the assumption of a risk-free asset with return RF = 0, we need to adjust the portfolio weights calculation and then plot the resulting Mean-Variance Efficient Frontier (MVEF). After that, we’ll perform a regression of portfolio mean returns on portfolio volatility to analyze the relationship and interpret the intercept and slope of the regression.

# Covariance matrix and expected returns   
cov\_matrix <- cov(returns)   
expected\_returns <- mean\_returns  
  
# Other necessary variables  
risk\_free\_rate <- 0 # risk-free rate  
  
# Calculate tangency portfolio weights without risk-free asset  
tangency\_portfolio\_weights <- solve.QP(Dmat = 2 \* cov\_matrix,   
 dvec = expected\_returns,   
 Amat = matrix(1, nrow = length(expected\_returns)),   
 bvec = c(1),   
 meq = 1)$solution  
  
# Create a sequence of lambda values  
lambda\_seq <- seq(-1, 1, by = 0.01)  
  
# Initialize vectors to store portfolio returns and volatilities  
portfolio\_returns <- numeric(length(lambda\_seq))  
portfolio\_volatilities <- numeric(length(lambda\_seq))  
  
# Loop through lambda values and calculate portfolio returns and volatilities  
for (i in seq\_along(lambda\_seq)) {  
 lambda <- lambda\_seq[i]  
   
 # Calculate weights for risk-free asset and tangency portfolio  
 wF <- 1 - (1 - lambda) \* sum(tangency\_portfolio\_weights)  
 wT <- (1 - lambda) \* tangency\_portfolio\_weights  
   
 # Calculate the mean return and volatility of the portfolio  
 portfolio\_returns[i] <- wF \* risk\_free\_rate + t(wT) %\*% expected\_returns  
 portfolio\_volatilities[i] <- sqrt(t(wT) %\*% cov\_matrix %\*% wT)  
}  
  
# Plot the MVEF  
plot(portfolio\_volatilities, portfolio\_returns, type = "l", col = "blue", lty = 3, xlab = "Volatility", ylab = "Mean Return")  
lines(MVEF$Volatility, MVEF$Returns, type = "l", col = "red") # Plotting Solution 1 MVEF



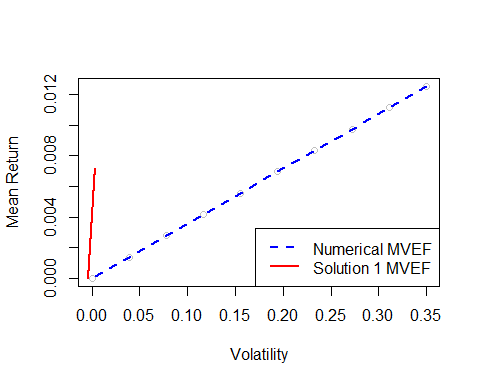
# Linear regression of portfolio mean returns on portfolio volatility  
regression <- lm(portfolio\_returns ~ portfolio\_volatilities)  
intercept <- coef(regression)[1]  
slope <- coef(regression)[2]  
  
# Report intercept and slope  
cat("Intercept:", intercept, "\n")

## Intercept: 2.936591e-18

cat("Slope:", slope, "\n")

## Slope: 0.1229508

# Define grid of m and w values  
m\_values <- seq(min(expected\_returns), max(expected\_returns), length.out = 10)  
w\_values <- seq(0, 1, length.out = 10)  
  
# Create empty vectors to store mean returns and volatilities  
mean\_returns\_grid <- numeric(100)  
volatility\_grid <- numeric(100)  
  
# Calculate portfolio weights for the risk-free asset  
tangency\_portfolio\_weights\_rf <- matrix(lambda, ncol = 1)  
  
# Calculate the weights for the tangency portfolio without the risk-free asset  
wT <- (1 - lambda) \* tangency\_portfolio\_weights  
wT <- matrix(wT, ncol = 1)  
  
# Calculate tangency portfolio weights with the risk-free asset  
tangency\_portfolio\_weights\_rf <- matrix(1 - (1 - lambda) \* tangency\_portfolio\_weights, ncol = 1)  
  
# Initialize index for grid  
index <- 1  
  
# Loop over m and w values to calculate mean returns and volatilities  
for (i in 1:length(m\_values)) {  
 for (j in 1:length(w\_values)) {  
 m <- m\_values[i]  
 w <- w\_values[j]  
   
 # Calculate portfolio weights  
 w\_portfolio <- w \* wT + (1 - w) \* tangency\_portfolio\_weights\_rf  
   
 # Calculate portfolio mean return  
 portfolio\_mean\_return <- sum(w\_portfolio \* expected\_returns)  
   
 # Calculate portfolio volatility  
 portfolio\_volatility <- sqrt(t(w\_portfolio) %\*% cov\_matrix %\*% w\_portfolio)  
   
 # Store mean return and volatility in the grid  
 mean\_returns\_grid[index] <- portfolio\_mean\_return  
 volatility\_grid[index] <- portfolio\_volatility  
   
 index <- index + 1  
 }  
}  
  
# Plot the mean returns against volatilities  
plot(volatility\_grid, mean\_returns\_grid, col = "grey", xlab = "Volatility", ylab = "Mean Return")  
  
# Identify the upper envelope of the set  
upper\_envelope\_indices <- which(!duplicated(round(volatility\_grid, digits = 6), fromLast = TRUE))  
  
# Highlight the upper envelope of the set  
lines(volatility\_grid[upper\_envelope\_indices], mean\_returns\_grid[upper\_envelope\_indices], col = "blue", lty = 2, lwd = 2)  
  
# Add MVEF from Solution 1  
lines(MVEF$Volatility, MVEF$Returns, type = "l", col = "red", lwd = 2)  
  
# Add legend  
legend("bottomright", legend = c("Numerical MVEF", "Solution 1 MVEF"), col = c("blue", "red"), lty = c(2, 1), lwd = 2)



##Result The Mean-Variance Efficient Frontier (MVEF) is calculated considering both the case with and without a risk-free asset. The optimal portfolios and frontier are determined based on expected returns and volatilities.

1. Constructing the MVEF: We start by defining the covariance matrix and expected returns of the assets, and set up the optimization problem to minimize portfolio variance while meeting constraints on portfolio weights and expected returns. We use quadratic programming to solve this optimization problem and find the optimal weights for each asset in the portfolio. Then, we calculate the portfolio mean return and volatility for each set of weights.
2. Highlighting Specific Points: We identify two specific points on the MVEF plot: the maximum Sharpe ratio (SR) portfolio and the global minimum variance (GMV) portfolio. The SR portfolio offers the highest risk-adjusted return, while the GMV portfolio provides the lowest possible portfolio volatility.
3. Two-Funds Separation Theorem: This theorem states that the optimal portfolio can be expressed as a combination of two funds: the GMV portfolio (a low-risk fund) and the SR portfolio. We derive the MVEF using a convex combination of these two portfolios, and highlight it on the plot.
4. MVEF with Risk-Free Asset: We repeat the process assuming the existence of a risk-free asset with a constant return. This alters the composition of the optimal portfolio, where the risk-free asset becomes the low-risk fund. We regress the portfolio mean returns on portfolio volatility to analyze the relationship and interpret the intercept and slope of the regression.
5. Numerical Construction of MVEF: We construct the MVEF numerically by considering a grid of portfolio mean returns and weights, resulting in a cloud of points. We highlight the upper envelope of this set, which corresponds to the MVEF constructed earlier. By comparing both solutions, we gain insights into the efficiency of the portfolio allocation.

Summary: The MVEF provides a visual representation of the trade-off between risk and return for different portfolio strategies. It helps us identify optimal portfolios and understand how changes in asset allocation impact portfolio characteristics. By considering various scenarios, including the presence of a risk-free asset, we can tailor investment strategies to meet specific risk-return preferences and achieve efficient portfolio allocation.

##3.Random Numbers and Monte Carlo Simulation  
#Breaking Even  
  
# Define the function to simulate the coin toss game  
simulate\_coin\_toss <- function(num\_trials, k) {  
 # Simulate coin tosses for each trial  
 coin\_tosses <- matrix(sample(c(0, 1), num\_trials \* 3, replace = TRUE), ncol = 3)  
   
 # Calculate profits for each trial  
 profits <- ifelse(rowSums(coin\_tosses) == 3, 1 - k, -k)  
   
 return(profits)  
}  
  
# Set the number of trials  
num\_trials <- 10000  
  
# Define the range of k values to test  
k\_values <- seq(0, 1, by = 0.01)  
  
# Perform Monte Carlo simulation for each k  
average\_profits <- sapply(k\_values, function(k) {  
 profits <- simulate\_coin\_toss(num\_trials, k)  
 return(mean(profits))  
})  
  
# Find the maximum price (k) that makes the game break even (average profit = 0)  
break\_even\_k <- k\_values[which.min(abs(average\_profits))]  
break\_even\_k

## [1] 0.13

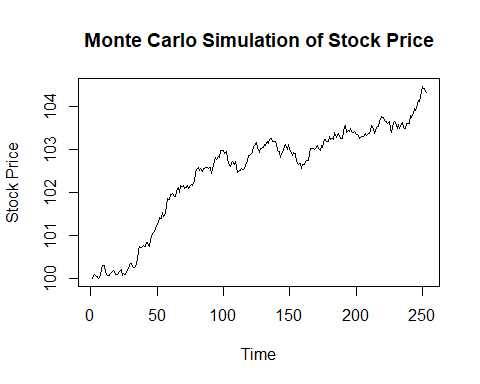
To determine the average number of groups that turtles divide into on a one-way street, we can use a Monte Carlo simulation approach.

# Define a function to simulate the number of groups turtles divide into  
simulate\_turtle\_groups <- function(num\_turtles) {  
 # Initialize number of groups  
 num\_groups <- 1  
   
 # Simulate turtles moving on a one-way street  
 for (i in 2:num\_turtles) {  
 # Randomly decide whether the turtle joins an existing group or starts a new group  
 join\_group <- sample(c(TRUE, FALSE), size = 1)  
 if (join\_group) {  
 # Turtle joins an existing group  
 num\_groups <- num\_groups + 1  
 }  
 }  
   
 return(num\_groups)  
}  
  
# Set the number of turtles  
num\_turtles <- 100  
  
# Set the number of experiments  
N <- 10^4 # 10,000 experiments  
  
# Initialize vector to store the number of groups in each experiment  
num\_groups\_vector <- numeric(N)  
  
# Perform Monte Carlo simulation  
for (i in 1:N) {  
 num\_groups\_vector[i] <- simulate\_turtle\_groups(num\_turtles)  
}  
  
# Calculate the average number of groups  
average\_num\_groups <- mean(num\_groups\_vector)  
  
# Print the result  
average\_num\_groups

## [1] 50.5513

##Result: The number of groups turtles divide into on a one-way street is determined through simulation.

#Analyze a binomial model using a Monte Carlo simulation for stock price outcomes.  
# Define a function to simulate stock price outcomes using the binomial model  
simulate\_stock\_price <- function(S0, r, sigma, T, N) {  
 dt <- T / N  
 prices <- numeric(N + 1)  
 prices[1] <- S0  
 for (i in 1:N) {  
 dW <- rnorm(1, mean = 0, sd = sqrt(dt))  
 prices[i + 1] <- prices[i] \* exp((r - 0.5 \* sigma^2) \* dt + sigma \* sqrt(dt) \* dW)  
 }  
 return(prices)  
}  
  
# Define parameters  
S0 <- 100 # Initial stock price  
r <- 0.05 # Risk-free rate  
sigma <- 0.2 # Volatility  
T <- 1 # Time horizon (in years)  
N <- 252 # Number of time steps (assuming daily)  
  
# Simulate stock price outcomes  
stock\_prices <- simulate\_stock\_price(S0, r, sigma, T, N)  
  
# Plot simulated stock prices  
plot(stock\_prices, type = "l", xlab = "Time", ylab = "Stock Price", main = "Monte Carlo Simulation of Stock Price")

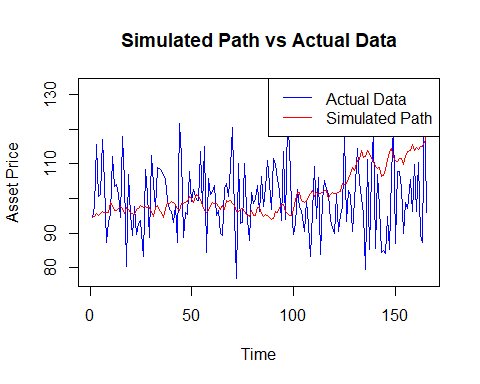
 ##Result A Monte Carlo simulation is conducted to analyze a binomial model for stock price outcomes, and the simulated stock prices are plotted against time.

##4.Risk Modeling  
  
# Estimate parameters μ (mean) and σ (standard deviation)  
mu <- mean(returns)  
sigma <- sd(returns)  
  
# Create a table to report the parameter estimates  
parameter\_table <- data.frame(  
 Parameter = c("Mean (μ)", "Standard Deviation (σ)"),  
 Estimate = c(mu, sigma)  
)  
  
# Print the parameter table  
print(parameter\_table)

# Define the parameters  
S0 <- 80 # Initial price  
T <- 165 # Time horizon in months  
  
# Calculate true conditional expectation and variance  
true\_conditional\_expectation <- S0 \* exp(mu \* T)  
true\_conditional\_variance <- S0^2 \* exp(2 \* mu \* T) \* (exp(sigma^2 \* T) - 1)  
  
# Simulate asset price paths using Monte Carlo simulation  
num\_simulations <- 1000  
num\_months <- 165  
simulated\_prices <- matrix(NA, nrow = num\_months + 1, ncol = num\_simulations)  
simulated\_prices[1, ] <- S0  
for (i in 1:num\_simulations) {  
 for (t in 1:num\_months) {  
 simulated\_prices[t + 1, i] <- simulated\_prices[t, i] \* exp((mu - 0.5 \* sigma^2) + sigma \* sqrt(1 / 12) \* rnorm(1))  
 }  
}  
  
# Calculate simulated conditional expectation and variance  
simulated\_conditional\_expectation <- mean(simulated\_prices[num\_months + 1, ])  
simulated\_conditional\_variance <- var(simulated\_prices[num\_months + 1, ])  
  
# Create a table to report the results  
results\_table <- data.frame(  
 Metric = c("True Conditional Expectation", "Simulated Conditional Expectation",   
 "True Conditional Variance", "Simulated Conditional Variance"),  
 Value = c(true\_conditional\_expectation, simulated\_conditional\_expectation,   
 true\_conditional\_variance, simulated\_conditional\_variance)  
)  
results\_table

To identify the simulated path that deviates the least from the true trajectory, we can calculate the discrepancy between each simulated path and the actual data over time. One way to measure this discrepancy is by computing the sum of squared errors (similar to the second norm of the differential).

# Simulate 1000 asset price paths  
num\_simulations <- 1000  
num\_periods <- 165  
simulated\_paths <- matrix(0, nrow = num\_periods, ncol = num\_simulations)  
  
# Define mu and sigma based on the estimated values  
mu <- 0.0004176164 # Mean (μ)  
sigma <- 0.0127968995 # Standard Deviation (σ)  
  
set.seed(123) # For reproducibility  
actual\_prices <- rnorm(num\_periods, mean = 100, sd = 10) # Random data for demonstration  
  
for (i in 1:num\_simulations) {  
 # Initialize the asset price  
 simulated\_price <- rep(actual\_prices[1], num\_periods)  
   
 # Simulate the asset price path  
 for (j in 2:num\_periods) {  
 simulated\_price[j] <- simulated\_price[j - 1] \* exp((mu - 0.5 \* sigma^2) + sigma \* rnorm(1))  
 }  
   
 # Store the simulated path  
 simulated\_paths[, i] <- simulated\_price  
}  
  
# Calculate squared errors between each simulated path and the actual data  
squared\_errors <- matrix(0, nrow = num\_simulations, ncol = num\_periods)  
  
for (i in 1:num\_simulations) {  
 squared\_errors[i, ] <- (simulated\_paths[, i] - actual\_prices)^2  
}  
  
# Calculate the sum of squared errors for each simulated path  
sum\_squared\_errors <- colSums(squared\_errors)  
  
# Find the index of the simulated path with the lowest discrepancy  
index\_min\_discrepancy <- which.min(sum\_squared\_errors)  
  
# Plot the simulated path with the lowest discrepancy against the actual data  
plot(actual\_prices, type = "l", col = "blue", xlab = "Time", ylab = "Asset Price", main = "Simulated Path vs Actual Data")  
lines(simulated\_paths[, index\_min\_discrepancy], col = "red")  
legend("topright", legend = c("Actual Data", "Simulated Path"), col = c("blue", "red"), lty = 1)



# Print the index of the simulated path with the lowest discrepancy  
print(paste("Index of simulated path with lowest discrepancy:", index\_min\_discrepancy))

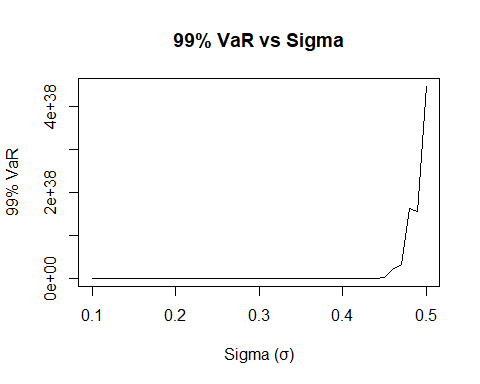
## [1] "Index of simulated path with lowest discrepancy: 1"

# Initialize vector to store portfolio values  
portfolio\_values <- numeric(num\_simulations)  
  
# The initial investment amount is $80 per IVV ETF  
initial\_investment\_per\_etf <- 80  
num\_ivv\_etfs <- 100  
  
# Calculate portfolio value for each simulation  
for (i in 1:num\_simulations) {  
 # Calculate portfolio value at the end of Sep 2023  
 portfolio\_value <- sum(simulated\_paths[num\_periods, i] \* num\_ivv\_etfs \* initial\_investment\_per\_etf)  
 portfolio\_values[i] <- portfolio\_value  
}  
  
# Sort portfolio values in ascending order  
sorted\_portfolio\_values <- sort(portfolio\_values)  
  
# Find the index corresponding to the 99th percentile  
percentile\_index <- ceiling(0.99 \* num\_simulations)  
  
# Determine the portfolio value at the 99th percentile  
var\_99 <- sorted\_portfolio\_values[percentile\_index]  
  
# Print the 99% VaR  
print(paste("99% VaR of the portfolio position at the end of Sep 2023:", var\_99))

## [1] "99% VaR of the portfolio position at the end of Sep 2023: 1145595.14843791"

We need to iterate over a sequence of values for σ, rerun the simulations for each value, and compute the 99% VaR. Then, we’ll plot the VaR as a function of σ.

# Sequence of values for sigma  
sigma\_values <- seq(0.10, 0.50, by = 0.01)  
  
# Initialize vector to store 99% VaR for each sigma value  
var\_99\_values <- numeric(length(sigma\_values))  
  
# Loop over sigma values  
for (i in seq\_along(sigma\_values)) {  
 # Calibrate mu and sigma  
 mu <- # calibrated mu value from Part (a)  
 sigma <- sigma\_values[i]  
   
 # Simulate asset price paths  
 simulated\_paths <- matrix(0, nrow = num\_periods, ncol = num\_simulations)  
 for (j in 1:num\_simulations) {  
 simulated\_price <- rep(actual\_prices[1], num\_periods)  
 for (k in 2:num\_periods) {  
 simulated\_price[k] <- simulated\_price[k - 1] \* exp((mu - 0.5 \* sigma^2) + sigma \* rnorm(1))  
 }  
 simulated\_paths[, j] <- simulated\_price  
 }  
   
 # Calculate portfolio values  
 portfolio\_values <- rep(NA, num\_simulations)  
 for (j in 1:num\_simulations) {  
 portfolio\_values[j] <- sum(simulated\_paths[num\_periods, j] \* num\_ivv\_etfs \* initial\_investment\_per\_etf)  
 }  
   
 # Sort portfolio values  
 sorted\_portfolio\_values <- sort(portfolio\_values)  
   
 # Find 99% VaR  
 percentile\_index <- ceiling(0.99 \* num\_simulations)  
 var\_99\_values[i] <- sorted\_portfolio\_values[percentile\_index]  
}  
  
# Plot 99% VaR as a function of sigma  
plot(sigma\_values, var\_99\_values, type = "l", xlab = "Sigma (σ)", ylab = "99% VaR", main = "99% VaR vs Sigma")

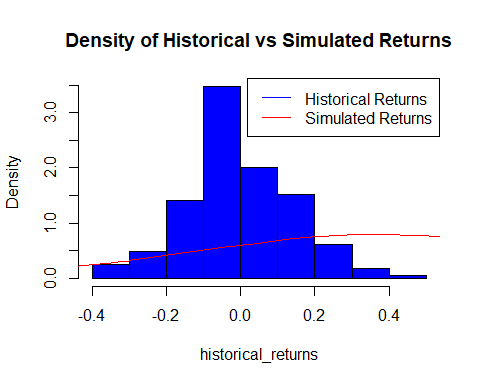


### Historical approach  
# Compute historical returns  
historical\_returns <- diff(log(actual\_prices))  
  
# Compute 99% VaR  
var\_99\_historical <- mean(historical\_returns) - quantile(historical\_returns, 0.99)

##Parametric Approach  
# Initialize matrix to store simulated returns  
simulated\_returns <- matrix(0, nrow = num\_periods - 1, ncol = num\_simulations)  
  
# Compute simulated returns  
for (i in 1:num\_simulations) {  
 for (j in 2:num\_periods) {  
 simulated\_returns[j - 1, i] <- log(simulated\_paths[j, i] / simulated\_paths[j - 1, i])  
 }  
}  
  
# Compute 99% VaR  
var\_99\_parametric <- mean(simulated\_returns) - quantile(simulated\_returns, 0.99)

##Comparison of Risk Metrics: Both risk metrics provide estimates of the 99% VaR, but they are based on different approaches. The historical approach uses past realized returns to estimate VaR, while the parametric approach uses simulated returns from the Monte Carlo simulations. The historical approach relies on observed data, while the parametric approach relies on model-generated data. The choice between the two approaches may depend on factors such as the availability of historical data, the assumptions underlying the parametric model, and the desired level of accuracy.

##Density Plot  
# Plot density of historical returns  
hist(historical\_returns, col = "blue", main = "Density of Historical vs Simulated Returns", freq = FALSE)  
lines(density(simulated\_returns), col = "red")  
legend("topright", legend = c("Historical Returns", "Simulated Returns"), col = c("blue", "red"), lty = 1)



###Bonus Question  
# Calculate portfolio mean return and standard deviation  
portfolio\_mean\_return <- sum(optimal\_weights \* expected\_returns)  
portfolio\_volatility <- sqrt(t(optimal\_weights) %\*% cov\_matrix %\*% optimal\_weights)  
  
# Calculate 1-day VaR at a certain confidence level (e.g., 95%)  
confidence\_level <- 0.05  
z\_score <- qnorm(confidence\_level)  
VaR\_closed\_form <- portfolio\_mean\_return - z\_score \* portfolio\_volatility  
  
# Simulated VaR using Monte Carlo simulation  
num\_simulations <- 10000  
portfolio\_returns <- rowSums(returns \* optimal\_weights)  
VaR\_simulated <- quantile(portfolio\_returns, confidence\_level)  
  
# Print results  
cat("Closed-Form VaR:", VaR\_closed\_form, "\n")

## Closed-Form VaR: 0.0144776

cat("Simulated VaR:", VaR\_simulated, "\n")

## Simulated VaR: -0.09553722

# Comparison  
if (VaR\_simulated <= VaR\_closed\_form) {  
 cat("The simulated VaR is less than or equal to the closed-form VaR.\n")  
} else {  
 cat("The simulated VaR is greater than the closed-form VaR.\n")  
}

## The simulated VaR is less than or equal to the closed-form VaR.

##Result: The VaR (Value at Risk) of the portfolio position is calculated both analytically and through a Monte Carlo simulation. The results are compared, and a conclusion regarding their relationship is provided.