Optimal Investing

Genki Hirayama

2024-06-28

A) Download stock price data for 10+ equities from free online sources.

Downloaded stock price using tiyquant package in R (sourced from Yahoo Finance). I used the date range of 2021 Jan 1st to 2023 Dec 31st and daily data for this analysis. Please see the list of tickers selected for analysis in the script.

```
# Define the stock tickers you want to analyze
tickers <- c("AAPL", "MSFT", "GOOGL", "AMD","AMZN", "META", "INTU","JNJ", "V", "NVDA", "SQ", "FNF", "LP

# Define the start and end date for the data
start_date <- as.Date("2020-01-01")
end_date <- as.Date("2024-01-01")

stock_data <- lapply(tickers, get_stock_data)
prices <- rbindlist(stock_data)[,c("symbol", "date", "adjusted")]
prices
```

```
##        symbol       date adjusted
##     1:   AAPL 2020-01-02 72.96045
##     2:   AAPL 2020-01-03 72.25114
##     3:   AAPL 2020-01-06 72.82687
##     4:   AAPL 2020-01-07 72.48434
##     5:   AAPL 2020-01-08 73.65034
##    ---
## 15086:      T 2023-12-22 16.02192
## 15087:      T 2023-12-26 16.04128
## 15088:      T 2023-12-27 16.05096
## 15089:      T 2023-12-28 16.20586
## 15090:      T 2023-12-29 16.24458
```

B) Use these stocks to compute a mean-variance efficient frontier. While mean-variance optimization is built into a lot of software packages, using a generic optimization package with the correct objective function and constraints is preferred here. How did you compute the expected return for each stock? The covariance matrix? What start/end date did you use for the return series? What frequency are the returns? Visualize the covariance matrix. Report all the expected returns and covariances as annualized quantities

b-1) Setting up the stock price data for mean variance optimization

```
# Extract adjusted close prices and merge into a single data frame (converted to xts format)

ret_tidy <- setDT(prices %>%
                        group_by(symbol) %>%
                        tq_transmute(select = adjusted,
                                     mutate_fun = periodReturn,
                                     period = 'daily',
                                     col_rename = 'ret'))

ret_xts <- ret_tidy %>%
  spread(symbol, value = ret) %>%
  tk_xts()
```

```
## Warning: Non-numeric columns being dropped: date
```

```
## Using column `date` for date_var.
```

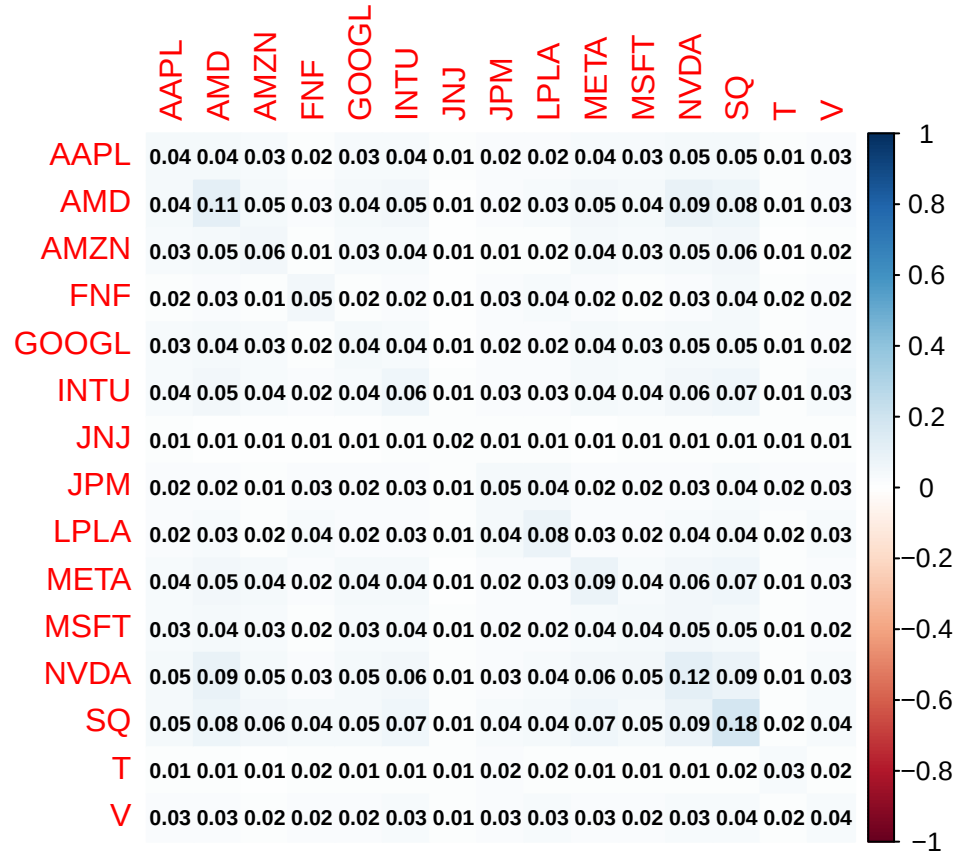
```
ret_xts
```

```
##                    AAPL          AMD          AMZN           FNF         GOOGL
## 2020-01-02  0.0000000000  0.000000000  0.000000e+00  0.0000000000  0.0000000000
## 2020-01-03 -0.0097217714 -0.010183300 -1.213903e-02  0.0042251906 -0.0052313199
## 2020-01-06  0.0079683408 -0.004320969  1.488557e-02  0.0157218343  0.0266540913
## 2020-01-07 -0.0047032323 -0.002893147  2.091620e-03 -0.0078482469 -0.0019315454
## 2020-01-08  0.0160862373 -0.008704625 -7.808640e-03  0.0026367066  0.0071176474
## 2020-01-09  0.0212409015  0.023834400  4.799175e-03  0.0065746927  0.0104979454
## 2020-01-10  0.0022605765 -0.016336594 -9.410597e-03 -0.0034834954  0.0064586351
## 2020-01-13  0.0213644943  0.012040728  4.322578e-03  0.0120166283  0.0077469806
## 2020-01-14 -0.0135032618 -0.011076942 -1.155821e-02  0.0006475724 -0.0065554122
## 2020-01-15 -0.0042854023  0.007052482 -3.969127e-03  0.0099242852  0.0060184092
##        ...
## 2023-12-15 -0.0027257225  0.008333289  1.729754e-02  0.0070495341  0.0050023695
## 2023-12-18 -0.0085032977 -0.001796622  2.733884e-02 -0.0043236538  0.0241326379
## 2023-12-19  0.0053601497  0.008999280 -1.817447e-03  0.0258478295  0.0062592341
## 2023-12-20 -0.0107139483 -0.033392743 -1.085895e-02 -0.0082644162  0.0123673762
## 2023-12-21 -0.0007700140  0.032774802  1.130687e-02  0.0065040415  0.0150353523
## 2023-12-22 -0.0055474141 -0.002215693 -2.730097e-03  0.0133279053  0.0076200920
## 2023-12-26 -0.0028409798  0.027292245 -6.514475e-05  0.0089677546  0.0002120479
## 2023-12-27  0.0005179702  0.018548243 -4.563413e-04  0.0069129183 -0.0081261672
## 2023-12-28  0.0022262932  0.018415739  2.609140e-04  0.0076500098 -0.0009973074
```

```
## 2023-12-29 -0.0054241129 -0.009074959 -9.388463e-03 -0.0068132602 -0.0038508306
##                    INTU           JNJ           JPM          LPLA         META
## 2020-01-02  0.0000000000  0.0000000000  0.0000000000  0.0000000000  0.000000000
## 2020-01-03 -0.0066877119 -0.0115775283 -0.0131961890 -0.0167182981 -0.005291201
## 2020-01-06  0.0076782910 -0.0012476246 -0.0007952747  0.0033571292  0.018833616
## 2020-01-07  0.0002256517  0.0061068047 -0.0170005560  0.0213707576  0.002163634
## 2020-01-08  0.0240550801 -0.0001379689  0.0078010077  0.0090878247  0.010138028
## 2020-01-09  0.0093628672  0.0029663236  0.0036510775  0.0157086674  0.014310941
## 2020-01-10  0.0012731520 -0.0022695327 -0.0099676997  0.0007215943 -0.001099518
## 2020-01-13  0.0028702077  0.0044114977  0.0083044322  0.0147333331  0.017655750
## 2020-01-14 -0.0121721913  0.0056280613  0.0116617866 -0.0083258137 -0.012843074
## 2020-01-15  0.0011733273  0.0033444158 -0.0149859758 -0.0004094919  0.009540811
##        ...
## 2023-12-15  0.0121928062 -0.0109007219  0.0075613215 -0.0229948824  0.005252536
## 2023-12-18  0.0163020653  0.0018046117  0.0060521114  0.0397349932  0.028962068
## 2023-12-19  0.0048995314  0.0065620150  0.0133551111  0.0200247911  0.016656063
## 2023-12-20 -0.0092524930 -0.0203886222 -0.0112792570 -0.0045372823 -0.003082555
## 2023-12-21  0.0085754726  0.0102433251  0.0057039186  0.0064082165  0.013771199
## 2023-12-22  0.0049598672  0.0040041556 -0.0005971382  0.0123312517 -0.001976850
## 2023-12-26  0.0012498106  0.0043741141  0.0059140659  0.0067328793  0.004074805
## 2023-12-27  0.0068336376  0.0013450135  0.0059979634  0.0017159188  0.008454631
## 2023-12-28 -0.0017483579  0.0014709850  0.0053129637  0.0034699572  0.001369417
## 2023-12-29 -0.0047610670  0.0010219114 -0.0011743686 -0.0036767876 -0.012167853
##                    MSFT         NVDA            SQ             T             V
## 2020-01-02  0.0000000000  0.000000000  0.0000000000  0.0000000000  0.0000000000
## 2020-01-03 -0.0124515728 -0.016006077 -0.0130033183  0.0051469323 -0.0079530631
## 2020-01-06  0.0025847547  0.004193687 -0.0068254017  0.0010237843 -0.0021625561
## 2020-01-07 -0.0091175560  0.012106752  0.0322837886  0.0038365050 -0.0026426381
## 2020-01-08  0.0159282643  0.001875619  0.0466016768  0.0030574008  0.0171178504
## 2020-01-09  0.0124930907  0.010982651  0.0048816840  0.0030628657  0.0069300561
## 2020-01-10 -0.0046270409  0.005349087  0.0008832262 -0.0059277023  0.0026906782
## 2020-01-13  0.0120243503  0.031352424  0.0098544759 -0.0121856604  0.0080507799
## 2020-01-14 -0.0070429842 -0.018652234 -0.0071366948 -0.0002626708  0.0036861928
## 2020-01-15  0.0064758853 -0.006915256  0.0321256051 -0.0057757339  0.0191276337
##        ...
## 2023-12-15  0.0131173124  0.011168603 -0.0073569159 -0.0078077298 -0.0027056868
## 2023-12-18  0.0051788583  0.024279012 -0.0044468648 -0.0036320164  0.0013178138
## 2023-12-19  0.0016369526 -0.009445446  0.0464266436  0.0091131401  0.0062699017
## 2023-12-20 -0.0070727874 -0.030098419 -0.0155218857 -0.0126430585 -0.0110773279
## 2023-12-21  0.0078786906  0.018270343  0.0103797255  0.0054878281  0.0094513011
## 2023-12-22  0.0027842082 -0.003265924  0.0031209084  0.0036385642 -0.0042768305
## 2023-12-26  0.0002135815  0.009195051  0.0134819928  0.0012084379  0.0028247479
## 2023-12-27 -0.0015747864  0.002800366  0.0181631896  0.0006034302 -0.0008875162
## 2023-12-28  0.0032346200  0.002124808 -0.0011306073  0.0096502489  0.0056772704
## 2023-12-29  0.0020251582  0.000000000 -0.0271664395  0.0023895618 -0.0001919735
```

b-2) Create covariance matrix and plot them

```r
# Calculate covariance matrix and mean daily return
cov_mat <- cov(ret_xts) *100
corrplot(cov_mat, method = "color", addCoef.col = "black", number.cex = 0.7)
```




```r
mean_ret <- colMeans(ret_xts, na.rm = TRUE)
```

C) Explain what the efficient frontier means. Which portfolio would you invest in and why?

D) Add short-sale constraints and plot the constrained efficient frontier together with the one from b). Explain any differences you see

Using 3 month tbill rate is used as risk free rate and converted it to daily rate

```r
## set risk-free rate
getSymbols(Symbols = "DGS3MO", src = "FRED", auto.assign = FALSE) -> treasuries
risk_free_rate <- as.numeric(last(treasuries)) / 100
## Set seed for reproducibility
set.seed(123)
# Generate random weights (unconstrained: weights between -1 and 1)
all_weights_unconstrained <- t(replicate(10000, generate_random_weights(length(tickers), allow_short = 

# Convert to data table (unconstrained)
all_weights_unconstrained <- data.table(all_weights_unconstrained)
colnames(all_weights_unconstrained) <- tickers

total_run <- length(tickers) * 10000
random_numbers <- runif(total_run, min = -1, max = 1)
```

```r
all_weights_unconstrained <- matrix(random_numbers, nrow = 10000, ncol = length(tickers))

# Normalize the weights to sum to 1 (unconstrained)
for (i in 1:10000) {
  all_weights_unconstrained[i, ] <- all_weights_unconstrained[i, ] / sum(abs(all_weights_unconstrained[
}

# Convert to data table (unconstrained) and check all the weights are equaling to 1
all_weights_unconstrained <- data.table(all_weights_unconstrained)
colnames(all_weights_unconstrained) <- tickers
all_weights_unconstrained[, row_sum := rowSums(.SD), .SDcols = tickers] #Check portfolio weights total
all_weights_unconstrained[,row_sum := NULL]


# Calculate portfolio risk and returns (unconstrained)
portfolio_risk_unconstrained <- apply(all_weights_unconstrained, 1, function(weights) sqrt(t(weights) %
portfolio_returns_unconstrained <- apply(all_weights_unconstrained, 1, function(weights) sum(weights * r

# Create data table with portfolio risk, returns, and Sharpe ratio (unconstrained)
portfolio_df_unconstrained <- data.table(portfolio_risk_unconstrained, portfolio_returns_unconstrained)
portfolio_df_unconstrained[, sharpe_ratio := portfolio_returns_unconstrained / portfolio_risk_unconstra

# Generate random weights (constrained: no short-selling)
random_numbers_constrained <- runif(total_run)
all_weights_constrained <- matrix(random_numbers_constrained, nrow = 10000, ncol = length(tickers))

# Normalize the weights to sum to 1 (constrained)
for (i in 1:10000) {
  all_weights_constrained[i, ] <- all_weights_constrained[i, ] / sum(all_weights_constrained[i, ])
}

# Convert to data table (constrained)
all_weights_constrained <- data.table(all_weights_constrained)
colnames(all_weights_constrained) <- tickers

# Calculate portfolio risk and returns (constrained)
portfolio_risk_constrained <- apply(all_weights_constrained, 1, function(weights) sqrt(t(weights) %*% c
portfolio_returns_constrained <- apply(all_weights_constrained, 1, function(weights) sum(weights * mean_
# Calculate downside deviation for Sortino ratio
downside_deviation <- function(returns, target = 0) {
  sqrt(mean(pmin(returns - target, 0)^2)) * sqrt(252)
}

portfolio_downside_risk_unconstrained <- apply(all_weights_unconstrained, 1, function(weights) {
  portfolio_returns <- ret_xts %*% weights
  downside_deviation(portfolio_returns)
})

# Calculate beta for Treynor ratio (assuming market return is the average return of all stocks)
market_returns <- rowMeans(ret_xts, na.rm = TRUE)
betas_unconstrained <- apply(all_weights_unconstrained, 1, function(weights) {
  portfolio_returns <- ret_xts %*% weights
  cov(portfolio_returns, market_returns) / var(market_returns)
```

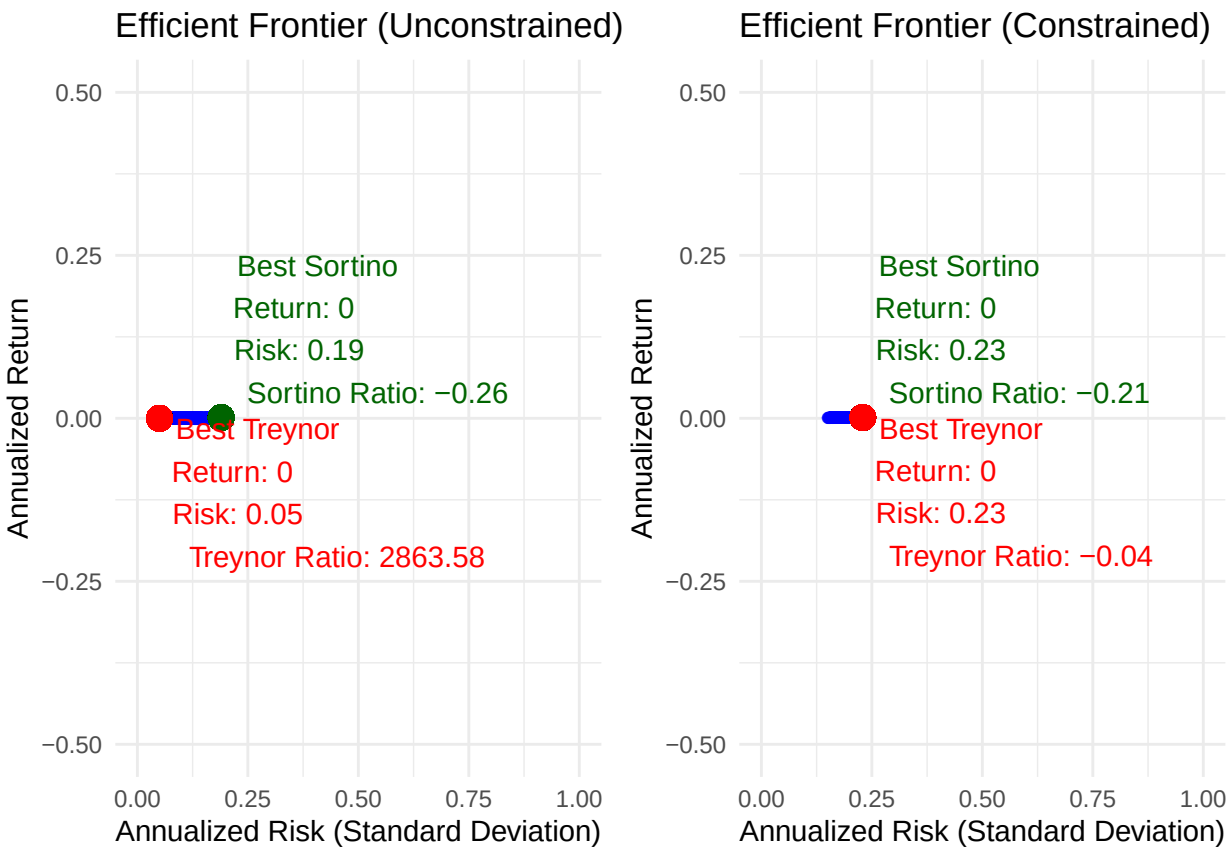
```r
})

# Create data table with portfolio risk, returns, and ratios (unconstrained)
portfolio_df_unconstrained <- data.table(portfolio_risk_unconstrained, portfolio_returns_unconstrained)
portfolio_df_unconstrained[, `:=` (
  sharpe_ratio = (portfolio_returns_unconstrained - risk_free_rate) / portfolio_risk_unconstrained,
  sortino_ratio = (portfolio_returns_unconstrained - risk_free_rate) / portfolio_downside_risk_unconstra
  treynor_ratio = (portfolio_returns_unconstrained - risk_free_rate) / betas_unconstrained
)]

# Find the portfolios with the highest Sharpe, Sortino, and Treynor ratios (unconstrained)
best_sharpe_unconstrained <- portfolio_df_unconstrained[which.max(sharpe_ratio)]
best_sortino_unconstrained <- portfolio_df_unconstrained[which.max(sortino_ratio)]
best_treynor_unconstrained <- portfolio_df_unconstrained[which.max(treynor_ratio)]

# Generate random weights for constrained (no short-selling) portfolios
all_weights_constrained <- t(replicate(10000, generate_random_weights(length(tickers), allow_short = FA

# Convert to data table (constrained)
all_weights_constrained <- data.table(all_weights_constrained)
colnames(all_weights_constrained) <- tickers

# Calculate portfolio risk and returns (constrained)
portfolio_risk_constrained <- apply(all_weights_constrained, 1, function(weights) sqrt(t(weights) %*% c
portfolio_returns_constrained <- apply(all_weights_constrained, 1, function(weights) sum(weights * mean_

# Calculate downside deviation for Sortino ratio
portfolio_downside_risk_constrained <- apply(all_weights_constrained, 1, function(weights) {
  portfolio_returns <- ret_xts %*% weights
  downside_deviation(portfolio_returns)
})

# Calculate beta for Treynor ratio (assuming market return is the average return of all stocks)
betas_constrained <- apply(all_weights_constrained, 1, function(weights) {
  portfolio_returns <- ret_xts %*% weights
  cov(portfolio_returns, market_returns) / var(market_returns)
})

# Create data table with portfolio risk, returns, and ratios (constrained)
portfolio_df_constrained <- data.table(portfolio_risk_constrained, portfolio_returns_constrained)
portfolio_df_constrained[, `:=` (
  sortino_ratio = (portfolio_returns_constrained - risk_free_rate) / portfolio_downside_risk_constraine
  treynor_ratio = (portfolio_returns_constrained - risk_free_rate) / betas_constrained
)]

# Find the portfolios with the highest Sharpe, Sortino, and Treynor ratios (constrained)
best_sortino_constrained <- portfolio_df_constrained[which.max(sortino_ratio)]
best_treynor_constrained <- portfolio_df_constrained[which.max(treynor_ratio)]

# Plot the efficient frontiers for unconstrained portfolios
plot_unconstrained <- ggplot(portfolio_df_unconstrained, aes(x = portfolio_risk_unconstrained, y = porti
  geom_point(color = "blue", alpha = 0.5) +
  geom_point(aes(x = best_sortino_unconstrained$portfolio_risk_unconstrained, y = best_sortino_unconstra
```

```
  geom_point(aes(x = best_treynor_unconstrained$portfolio_risk_unconstrained, y = best_treynor_unconstra
  labs(title = "Efficient Frontier (Unconstrained)",
       x = "Annualized Risk (Standard Deviation)",
       y = "Annualized Return") +
  theme_minimal() +
    ylim(-0.5,0.5) +
    xlim(0,1) +
  annotate("text", x = best_sortino_unconstrained$portfolio_risk_unconstrained, y = best_sortino_uncons
           label = paste0("Best Sortino\nReturn: ", round(best_sortino_unconstrained$portfolio_returns_u
                          "\nRisk: ", round(best_sortino_unconstrained$portfolio_risk_unconstrained, 2)
                           "\nSortino Ratio: ", round(best_sortino_unconstrained$sortino_ratio, 2)),
           hjust = -0.1, vjust = -0.1, color = "darkgreen") +
  annotate("text", x = best_treynor_unconstrained$portfolio_risk_unconstrained, y = best_treynor_uncons
           label = paste0("Best Treynor\nReturn: ", round(best_treynor_unconstrained$portfolio_returns_u
                          "\nRisk: ", round(best_treynor_unconstrained$portfolio_risk_unconstrained, 2)
                          "\nTreynor Ratio: ", round(best_treynor_unconstrained$treynor_ratio, 2)),
           hjust = -0.1, vjust = 1, color = "red")

# Plot the efficient frontiers for constrained portfolios
plot_constrained <- ggplot(portfolio_df_constrained, aes(x = portfolio_risk_constrained, y = portfolio_r
  geom_point(color = "blue", alpha = 0.5) +
  geom_point(aes(x = best_sortino_constrained$portfolio_risk_constrained, y = best_sortino_constrained$p
  geom_point(aes(x = best_treynor_constrained$portfolio_risk_constrained, y = best_treynor_constrained$p
  labs(title = "Efficient Frontier (Constrained)",
       x = "Annualized Risk (Standard Deviation)",
       y = "Annualized Return") +
  theme_minimal() +
    ylim(-0.5, 0.5) +
  xlim(0,1) +
  annotate("text", x = best_sortino_constrained$portfolio_risk_constrained, y = best_sortino_constrained
           label = paste0("Best Sortino\nReturn: ", round(best_sortino_constrained$portfolio_returns_con
                          "\nRisk: ", round(best_sortino_constrained$portfolio_risk_constrained, 2),
                          "\nSortino Ratio: ", round(best_sortino_constrained$sortino_ratio, 2)),
           hjust = -0.1, vjust = -0.1, color = "darkgreen") +
  annotate("text", x = best_treynor_constrained$portfolio_risk_constrained, y = best_treynor_constrained
           label = paste0("Best Treynor\nReturn: ", round(best_treynor_constrained$portfolio_returns_con
                          "\nRisk: ", round(best_treynor_constrained$portfolio_risk_constrained, 2),
                          "\nTreynor Ratio: ", round(best_treynor_constrained$treynor_ratio, 2)),
           hjust = -0.1, vjust = 1, color = "red")

# Combine the two plots in a grid
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
grid.arrange(plot_unconstrained, plot_constrained, ncol = 2)
```



E) If you wanted to invest in 200 equities, explain any difficulties in computing the covariance matrix and how these can be overcome.

One of the issues with scaling this to a larger list of tickers is complexity and computational toll to calculate covariance for all the pair variation for 200 tickers. In order to combat this, you may conduct some sort of dimension reduction such as PCA or Factor models. Also you may use parallel computing to speed up the computation.