Coursework1

202353542 - Nisar Ahmed

2024-02-24

Part 1 - Optimization of the Portfolio using GA

Checking and installing necessary packages and libraries

Ensuring all necessary R packages are installed and loaded for portfolio analysis and optimization.

```
necessary_packages <- c("GA", "quantmod", "TTR", "xts", "zoo", "PerformanceAnalytics", "dplyr", "reshape
for(package in necessary_packages) {
  if (!requireNamespace(package, quietly = TRUE)) {
    install.packages(package)
  }
}
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(GA)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Package 'GA' version 3.2.4
## Type 'citation("GA")' for citing this R package in publications.

##
## Attaching package: 'GA'

## The following object is masked from 'package:utils':
##
##     de
```

```
library(quantmod)
```

```
## Loading required package: xts

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: TTR
```

```r
library(xts)
library(zoo)
library(TTR)
library(PerformanceAnalytics)
```

```
##
## Attaching package: 'PerformanceAnalytics'

## The following object is masked from 'package:graphics':
##
##     legend
```

```r
library(dplyr)
```

```
##
## ######################### Warning from 'xts' package ##########################
## #                                                                             #
## # The dplyr lag() function breaks how base R's lag() function is supposed to  #
## # work, which breaks lag(my_xts). Calls to lag(my_xts) that you type or       #
## # source() into this session won't work correctly.                            #
## #                                                                             #
## # Use stats::lag() to make sure you're not using dplyr::lag(), or you can add #
## # conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop           #
## # dplyr from breaking base R's lag() function.                                #
## #                                                                             #
## # Code in packages is not affected. It's protected by R's namespace mechanism #
## # Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
## #                                                                             #
## ###############################################################################

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:xts':
##
##     first, last

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(ggplot2)
library(reshape2)
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

## The following object is masked from 'package:reshape2':
##
##     smiths
```

Part 1(a): Selection of Assets

Selecting 10 diverse assets from different sectors to construct a well-rounded portfolio. This diversity aims to mitigate risk by spreading investments across diff market behaviors, including tech, healthcare, and energy sectors.

```
my_assets <- c("AAPL", "PFE", "BAC", "TSLA", "PG", "XOM", "NEE", "BA", "DD", "SPG")
```

Part 1(b): Data Retrieval and Pre-processing

Fetching historical price data for the selected assets from 2020 to 2022.

Why COVID-19 Period?

- The three-year period from 2020 to 2022 is strategically chosen to encompass the market fluctuations due to the COVID-19 pandemic.This timeframe allows for the analysis of assets' performance through significant economic disruptions, providing insights into their resilience and potential for recovery.

Why slecting three years?

- Span of three years offers a balance between capturing recent market behaviors and ensuring enough data for meaningful analysis and visualization.The inclusion of this volatile period is crucial for optimizing a portfolio that can withstand and capitalize on market upheavals, making the analysis more relevant for current and future investing environments.

```
asset_prices <- list()
for(asset in my_assets) {
  getSymbols(asset, from = "2020-01-01", to = "2023-12-31", auto.assign = TRUE)
  asset_prices[[asset]] <- Cl(get(asset))
}
combined_prices <- do.call(merge, asset_prices)
combined_prices <- na.omit(combined_prices)
```

Calculating and visualizing daily returns

Combining and cleaning the asset price data to ensure a consistent and complete dataset for analysis. Daily returns are calculated to understand the assets' day-to-day performance, crucial for portfolio optimization.

```
daily_returns <- ROC(combined_prices, type = "discrete")
daily_returns <- na.omit(daily_returns)
myRetData <- daily_returns
```

Part 1(d): Portfolio Optimization Using Genetic Algorithm

Defining a fitness function to maximize the portfolio's Sharpe Ratio, balancing risk and return

The Sharpe Ratio is chosen as it is a widely used measure for calculating risk-adjusted return, enhancing portfolio performance.

```
fitness_function_alpha <- function(weights, alpha=0.5) {
  normalizedWeights <- weights / sum(weights)
  portfolioReturn <- sum(colMeans(myRetData, na.rm = TRUE) * normalizedWeights) * 252
  portfolioRisk <- sqrt(t(as.matrix(normalizedWeights)) %*% cov(myRetData) %*% as.matrix(normalizedWeig
  SharpeRatio <- portfolioReturn / portfolioRisk
  return(SharpeRatio) # The goal is to maximize the Sharpe Ratio
}
```

Running the genetic algorithm to find optimal portfolio weights

- Configuring and running the genetic algorithm with specified parameters to find the optimal asset weights
- Parameters such as population size and maximum iterations are tuned to efficiently search the solution space while balancing computational resources

```
ga_result <- ga(type = "real-valued",
                fitness = function(weights) fitness_function_alpha(weights, alpha=0.5),
                lower = rep(0, length(my_assets)),
                upper = rep(1, length(my_assets)),
                popSize = 50,
                maxiter = 100)

optimal_weights <- ga_result@solution
normalized_optimal_weights <- optimal_weights / sum(optimal_weights)
```

Detailed Evaluation for Future Performance

- Splitting the dataset into training and testing sets for a fair comparison
- Splitting the dataset to train and test the portfolio, ensuring a robust evaluation of its future performance - historical data informing future decisions.

```
split_date <- as.Date("2023-01-01")
training_data <- daily_returns[index(daily_returns) < split_date]
testing_data <- daily_returns[index(daily_returns) >= split_date]

# Re-optimizing the portfolio using only the training data
ga_result_training <- ga(type = "real-valued",
                         fitness = function(weights) fitness_function_alpha(weights, alpha=0.5),
                         lower = rep(0, length(my_assets)),
                         upper = rep(1, length(my_assets)),
                         popSize = 50,
                         maxiter = 100,
                         suggestions = normalized_optimal_weights)

optimal_weights_training <- ga_result_training@solution
normalized_optimal_weights_training <- optimal_weights_training / sum(optimal_weights_training)
```

Part 1(e): Comparison with Other Portfolios

Balanced Portfolio

- Comparing the GA-optimized portfolio against balanced and randomly generated portfolios to evaluate its relative performance. This step assesses the effectiveness of the genetic algorithm in enhancing portfolio returns and reducing risk.

```
balanced_weights <- rep(1 / length(my_assets), length(my_assets))
```

Generating several random portfolios for a broader comparison

```
set.seed(123) # Ensuring reproducibility
random_weights_list <- replicate(100, runif(length(my_assets)))
random_weights_list <- apply(random_weights_list, 2, function(x) x / sum(x))

print("Optimal Weights:")
```

```
## [1] "Optimal Weights:"
```

```r
print(optimal_weights)
```

```
##             x1        x2         x3        x4        x5       x6        x7
## [1,] 0.7795186 0.1523568 0.08547306 0.8786081 0.2226914 0.390743 0.1331852
##              x8         x9        x10
## [1,] 0.03662279 0.06996158 0.02661199
```

```r
print("Normalized Optimal Weights:")
```

```
## [1] "Normalized Optimal Weights:"
```

```r
print(normalized_optimal_weights)
```

```
##             x1         x2         x3        x4         x5        x6         x7
## [1,] 0.2808294 0.05488806 0.03079253 0.3165274 0.08022681 0.1407691 0.04798131
##              x8         x9         x10
## [1,] 0.01319373 0.02520436 0.009587237
```

```r
balanced_weights <- rep(1 / length(my_assets), length(my_assets))
print("Balanced Portfolio Weights:")
```

```
## [1] "Balanced Portfolio Weights:"
```

```r
print(balanced_weights)
```

```
##  [1] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
```

```r
print("Sample Random Portfolio Weights:")
```

```
## [1] "Sample Random Portfolio Weights:"
```

```r
print(random_weights_list[, 1])
```

```
##  [1] 0.049732600 0.136326595 0.070726967 0.152705787 0.162640959 0.007878374
##  [7] 0.091328624 0.154331672 0.095363146 0.078965276
```

Part 1(e): Evaluating Portfolio Performance - Evaluation of Optimized, Balanced, and Random Portfolios

- In this section, we evaluated the performance of optimized, balanced, and randomly generated portfolios.
- We defined a function to calculate annualized return, risk, and the Sharpe Ratio for each portfolio strategy using test data.
- This analysis helps us understand the effectiveness of genetic algorithm optimization in achieving superior risk-adjusted returns compared to simpler portfolio construction methods.

```r
## Defining the updated calculate_performance function
calculate_performance <- function(weights, returns) {
  weights_vector <- as.numeric(weights) # Ensuring weights are a numeric vector

  # Converting returns to an xts object if not already done
  if(!is.xts(returns)) {
    returns <- as.xts(returns, order.by=index(returns))
  }

  # Calculating portfolio returns using the provided weights
  portfolio_returns <- Return.portfolio(R = returns, weights = weights_vector, rebalance_on = "years")

  # Calculating annualized return, annualized risk and Sharpe Ratio
```

```r
  annualized_return <- annualReturn(portfolio_returns, scale = 252)
  annualized_risk <- sd(portfolio_returns) * sqrt(252)
  sharpe_ratio <- SharpeRatio.annualized(portfolio_returns, Rf = 0, scale = 252)
  return(c(annualized_return = as.numeric(annualized_return),
           annualized_risk = annualized_risk,
           sharpe_ratio = as.numeric(sharpe_ratio)))
}

# Using the performance calculation function to evaluate the optimized portfolio on test data
optimized_performance <- calculate_performance(normalized_optimal_weights_training, testing_data)

# Evaluating the performance of a balanced portfolio, where each asset is equally weighted, on test dat
balanced_performance <- calculate_performance(balanced_weights, testing_data)

# Applying the performance calculation across several randomly generated portfolios to assess their ave
random_performances <- apply(random_weights_list, 2, function(weights) calculate_performance(weights, te

print("Optimized Portfolio Performance:")
```

```
## [1] "Optimized Portfolio Performance:"
```

```r
print(optimized_performance)
```

```
## annualized_return   annualized_risk      sharpe_ratio
##        -0.8158694         0.3256838         1.7452432
```

```r
print("Balanced Performance:")
```

```
## [1] "Balanced Performance:"
```

```r
print(balanced_performance)
```

```
## annualized_return   annualized_risk      sharpe_ratio
##        -0.6121907         0.1668092         0.8348465
```

```r
print("Random Performance:")
```

```
## [1] "Random Performance:"
```

```r
print(random_performances)
```

```
##                         [,1]       [,2]       [,3]       [,4]       [,5]
## annualized_return -0.6068459 -0.6679049 -0.7271959 -0.6793694 -0.3025030
## annualized_risk    0.1903011  0.1797101  0.1909382  0.1922130  0.1879799
## sharpe_ratio       0.9272363  1.0304200  1.0260963  0.7845232  0.8932604
##                         [,6]       [,7]      [,8]       [,9]      [,10]
## annualized_return -2.3961028 -0.5226347 0.1175974 -0.5913188 -0.5844816
## annualized_risk    0.1593286  0.1494892 0.1367038  0.1918842  0.1982757
## sharpe_ratio       0.5247557  0.9300300 0.4379962  0.7455569  0.5016415
##                        [,11]      [,12]      [,13]      [,14]      [,15]
## annualized_return -0.7197836 -0.7113566 -0.7047535 -0.6076040 -0.5907556
## annualized_risk    0.1931991  0.1976673  0.1518145  0.1691613  0.1401800
## sharpe_ratio       0.9656786  1.4629477  0.7658906  0.7252527  0.5648326
##                        [,16]      [,17]      [,18]      [,19]      [,20]
## annualized_return -0.6490644 -0.6120230 -0.6443772 -0.4636507 -0.5602856
## annualized_risk    0.1537595  0.1619708  0.1808835  0.1539910  0.1760668
## sharpe_ratio       0.7193256  0.5689555  1.1399881  1.0346146  0.8877407
```

```
##                         [,21]      [,22]      [,23]     [,24]      [,25]
## annualized_return -0.6845052 -0.6471202 -0.4753004 0.2591189 -0.6195343
## annualized_risk    0.1689055  0.1930943  0.1879413 0.1533031  0.1727581
## sharpe_ratio       0.2541758  1.1728564  0.6922214 0.8827580  1.0149230
##                         [,26]      [,27]      [,28]      [,29]      [,30]
## annualized_return -0.5070314 -0.7055220 -0.6713714 -0.6295926 -0.7074795
## annualized_risk    0.1601017  0.1966671  0.1424516  0.1721790  0.2002129
## sharpe_ratio       0.7771884  0.9954314  0.8068343  1.4076158  0.8703002
##                         [,31]       [,32]      [,33]      [,34]      [,35]
## annualized_return -0.7149958 -0.20756269 -0.6319526 -0.5852803 -0.3555951
## annualized_risk    0.1977032  0.14858899  0.2057283  0.1909957  0.1559495
## sharpe_ratio       1.3350590  0.06795531  1.0748274  1.1074927  0.5476396
##                         [,36]      [,37]      [,38]      [,39]      [,40]
## annualized_return -0.3941377 -0.5790928 -0.5558852 -0.7541994 -0.6561782
## annualized_risk    0.1439702  0.1731690  0.1644331  0.1887503  0.2136972
## sharpe_ratio      -0.1330123  0.5132930  0.5589912  0.8432379  1.3477954
##                         [,41]      [,42]      [,43]      [,44]         [,45]
## annualized_return -0.6336101 -0.6199503 -0.5672422 -0.7649532 -0.4302657024
## annualized_risk    0.1784910  0.1749871  0.1480848  0.2219810  0.1413421290
## sharpe_ratio       1.0586098  0.2717414  0.6311397  1.3626626 -0.0002491627
##                         [,46]      [,47]      [,48]      [,49]      [,50]
## annualized_return -0.6467602 -0.4535563 -0.6283638 -0.5674961 -0.7187629
## annualized_risk    0.1445642  0.1476000  0.1934532  0.1818306  0.1919411
## sharpe_ratio       0.2865127  1.1260294  0.4421826  0.9228489  1.3125305
##                       [,51]      [,52]      [,53]      [,54]      [,55]
## annualized_return 0.5893686 -0.3180788 -0.5320080 -0.7058621 -0.6477856
## annualized_risk   0.1483920  0.1706795  0.1424952  0.1902795  0.1534023
## sharpe_ratio      0.5041963  1.1118659 -0.1679830  1.1737934  1.1232229
##                         [,56]     [,57]      [,58]      [,59]      [,60]
## annualized_return -0.7125458 0.5246862 -0.6715807 -0.4525988 -0.5608480
## annualized_risk    0.2041033 0.1523999  0.1476653  0.1516764  0.1605536
## sharpe_ratio       1.0099835 0.2215221  0.3398034  0.8498039  0.5408482
##                         [,61]      [,62]      [,63]      [,64]      [,65]
## annualized_return -0.6194963 -0.7498784 -0.5387696 -0.5823691 -0.5288897
## annualized_risk    0.1643110  0.1952781  0.1656493  0.1817494  0.1644988
## sharpe_ratio       0.6613691  1.1993129  0.9089995  0.8482503  0.3306563
##                         [,66]      [,67]       [,68]      [,69]      [,70]
## annualized_return -0.6972248 -0.6873483 -0.56568927 -0.7051909 -0.4580275
## annualized_risk    0.1554105  0.1906292  0.15487070  0.1821596  0.1533714
## sharpe_ratio       0.3697975  0.7686142  0.01198069  0.8222788 -0.1067415
##                         [,71]      [,72]      [,73]      [,74]      [,75]
## annualized_return -0.7500484 -0.7237352 -0.6792915 -0.6359889 -0.6778719
## annualized_risk    0.2062887  0.1963841  0.1810809  0.2042439  0.1777224
## sharpe_ratio       1.3103677  1.0436377  0.6888381  0.8281757  1.1657337
##                         [,76]      [,77]      [,78]      [,79]      [,80]
## annualized_return -0.3168627 -0.5629994 -0.6216409 -0.6761671 -0.7568533
## annualized_risk    0.1457241  0.1771930  0.1932016  0.1817336  0.1625329
## sharpe_ratio       0.5568552  1.2201127  1.0845567  0.6784605  1.1614268
##                         [,81]      [,82]      [,83]      [,84]       [,85]
## annualized_return -0.5582750 -0.6551968 -0.6825083 -0.6549494 -0.61480091
## annualized_risk    0.1462529  0.2123171  0.2162138  0.1841384  0.13644809
## sharpe_ratio       0.4372254  0.7884073  1.1927885  0.4811404  0.07290434
##                         [,86]      [,87]      [,88]      [,89]      [,90]
## annualized_return -0.5284442 -0.6294524 -0.5691407 -0.6238142 -0.6565163
```

```
## annualized_risk    0.1860328  0.1818047  0.1729210  0.1388181  0.1656535
## sharpe_ratio       0.4818124  1.1764240  1.0286194 -0.3501281  0.8202817
##                       [,91]      [,92]      [,93]      [,94]      [,95]
## annualized_return -0.6341604 -0.6140207 -0.7154729 -0.6494174 -0.6577706
## annualized_risk    0.1623150  0.1889405  0.1948622  0.1679987  0.1745111
## sharpe_ratio       0.9206602  0.7294112  1.4276761  1.2269547  1.1560271
##                       [,96]      [,97]      [,98]      [,99]     [,100]
## annualized_return -0.5351004 -0.5500218 -0.6008500 -0.6839405 -0.6533890
## annualized_risk    0.1490595  0.1373488  0.1593712  0.1711348  0.1634017
## sharpe_ratio       0.2160212  0.2947455  0.6235499  0.6635717  0.4657341
```

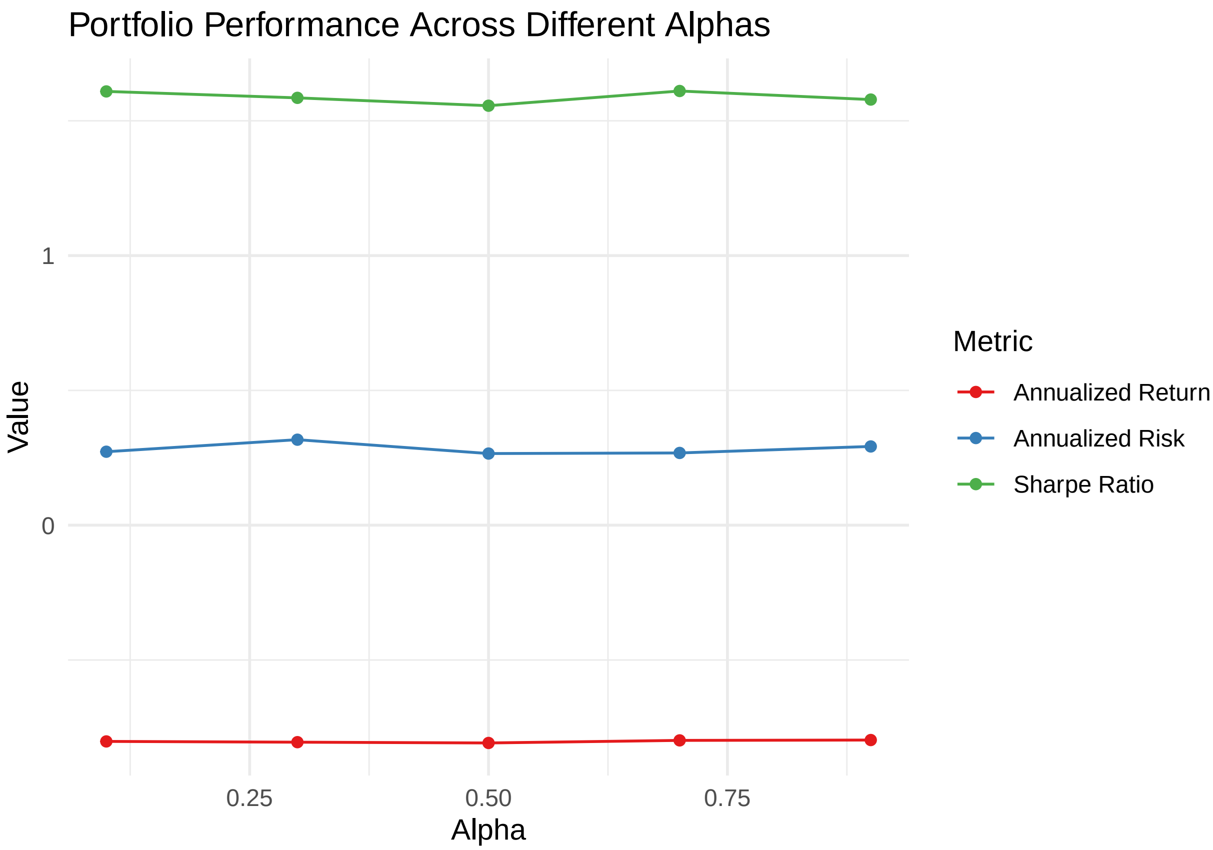
Part 1(f) Exploring Different Weightings of Risk and Return

- Adjusting the fitness function to explore portfolios with varying preferences for risk and return.
- By altering the alpha parameter, we simulated different investor profiles, from risk-averse (lower alpha) to return-seeking (higher alpha), analyzing how these preferences impact portfolio composition and performance.

```
alphas <- seq(0.1, 0.9, by = 0.2)
alpha_performances <- sapply(alphas, function(alpha) {
  ga_result_alpha <- ga(type = "real-valued",
                        fitness = function(weights) fitness_function_alpha(weights, alpha = alpha),
                        lower = rep(0, length(my_assets)),
                        upper = rep(1, length(my_assets)),
                        popSize = 50,
                        maxiter = 100)
  optimal_weights_alpha <- ga_result_alpha@solution / sum(ga_result_alpha@solution)
  calculate_performance(optimal_weights_alpha, testing_data)
})
```

Portfolio Performance Across Different Alphas

```
# Converting the matrix to a data frame for easier plotting with ggplot2
performance_df <- as.data.frame(t(alpha_performances))
colnames(performance_df) <- c("Annualized Return", "Annualized Risk", "Sharpe Ratio")
performance_df$Alpha <- alphas

# Melting the data frame for use with ggplot2
melted_performance_df <- melt(performance_df, id.vars = "Alpha", variable.name = "Metric", value.name =

# Plotting
ggplot(melted_performance_df, aes(x = Alpha, y = Value, color = Metric)) +
  geom_line() +
  geom_point() +
  theme_minimal() +
  labs(title = "Portfolio Performance Across Different Alphas",
       x = "Alpha",
       y = "Value",
       color = "Metric") +
  scale_color_brewer(palette = "Set1")
```



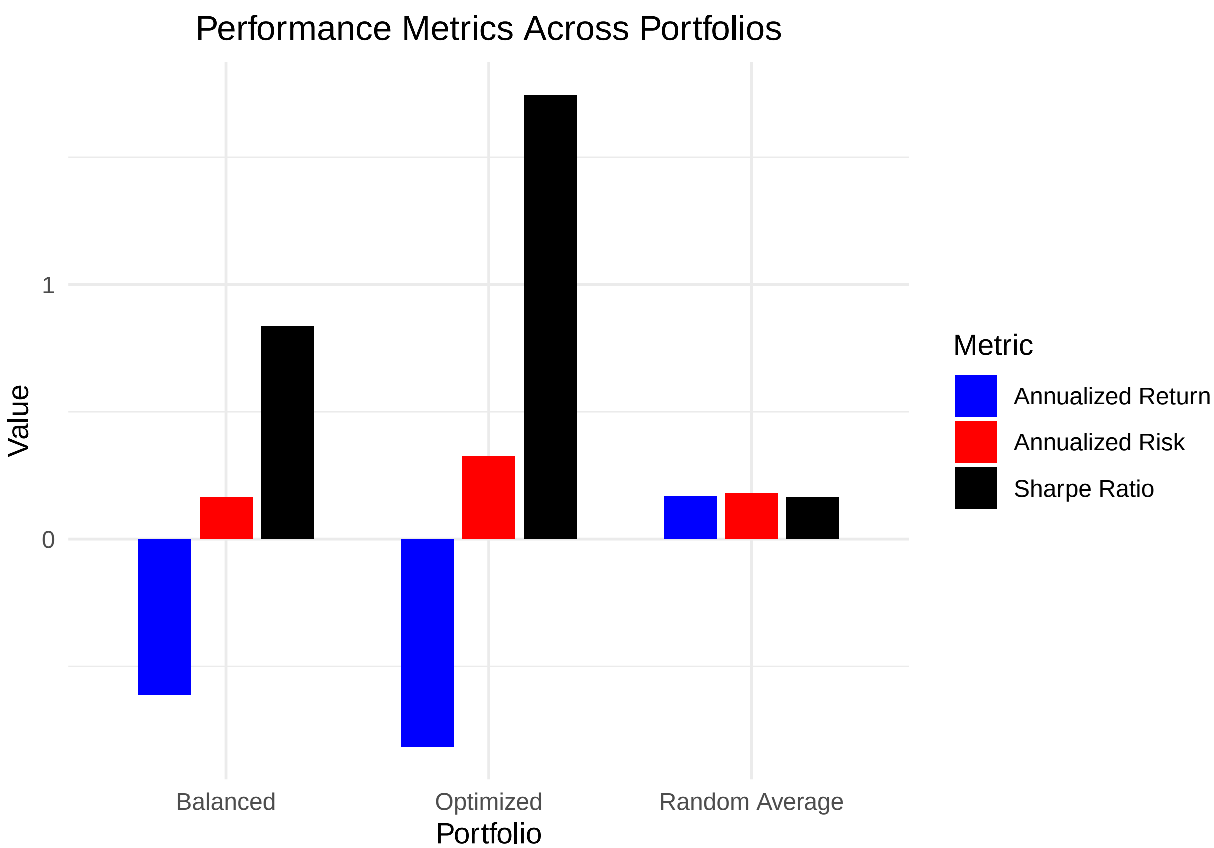
-

The graph shows how portfolio performance metrics change with different levels of alpha. Alpha adjusts the importance of risk versus return in our strategy. - As we slide the alpha from 0 to 1, we notice shifts in annualized return (red), risk (blue), and the Sharpe Ratio (green). It's clear that the balance between seeking returns and managing risk significantly affects our portfolio's performance.

Part 1(g) Visualization

```
# Converting random performance metrics to a matrix if they are in list format
if(is.list(random_performances)) {
  random_performances_matrix <- do.call(cbind, random_performances)
  random_average_metrics <- colMeans(random_performances_matrix)
} else {
  random_average_metrics <- colMeans(random_performances)
}

# Ensuring that we only take the first three metrics (Annualized Return, Annualized Risk, Sharpe Ratio)
random_average_metrics <- random_average_metrics[1:3]

# Retrieving the performance metrics for the optimized and balanced portfolios
optimized_metrics <- optimized_performance
balanced_metrics <- balanced_performance

# Combining all performance metrics into one matrix for comparison
performance_metrics_matrix <- rbind(optimized_metrics, balanced_metrics, random_average_metrics)
rownames(performance_metrics_matrix) <- c("Optimized", "Balanced", "Random Average")
colnames(performance_metrics_matrix) <- c("Annualized Return", "Annualized Risk", "Sharpe Ratio")
```

```
# Transforming the data into a long format for ggplot2 without using rownames_to_column
performance_long <- as.data.frame(performance_metrics_matrix)
performance_long$Portfolio <- rownames(performance_long)
performance_long <- reshape2::melt(performance_long, id.vars = "Portfolio")
colnames(performance_long) <- c("Portfolio", "Metric", "Value")

# Plotting
ggplot(performance_long, aes(x = Portfolio, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.7), width = 0.6) +
  scale_fill_manual(values = c("Annualized Return" = "blue", "Annualized Risk" = "red", "Sharpe Ratio" =
  theme_minimal() +
  labs(title = "Performance Metrics Across Portfolios", x = "Portfolio", y = "Value") +
  theme(plot.title = element_text(hjust = 0.5))  # Center the plot title
```



- The above graph displays the performance metrics for three different portfolio strategies: Balanced, Optimized, and Random Average, where each bar is representing a metric (red for Annualized Return, blue for Annualized Risk, and black for Sharpe Ratio).
- As it is evident that all portfolios are showing a negative Sharpe Ratio, indicating that the risk-adjusted returns are below expectations - the negative Sharpe Ratios across all portfolios could potentially be attributed to the volatility and unpredictable market conditions during the COVID-19 period.
- The Optimized Portfolio's lower risk suggests an attempt to mitigate this volatility, but the persisting negative returns indicate that the broader market challenges during the COVID-19 era likely had an overarching impact on portfolio performance.

Part 2(a): Data Fetching and Preprocessing

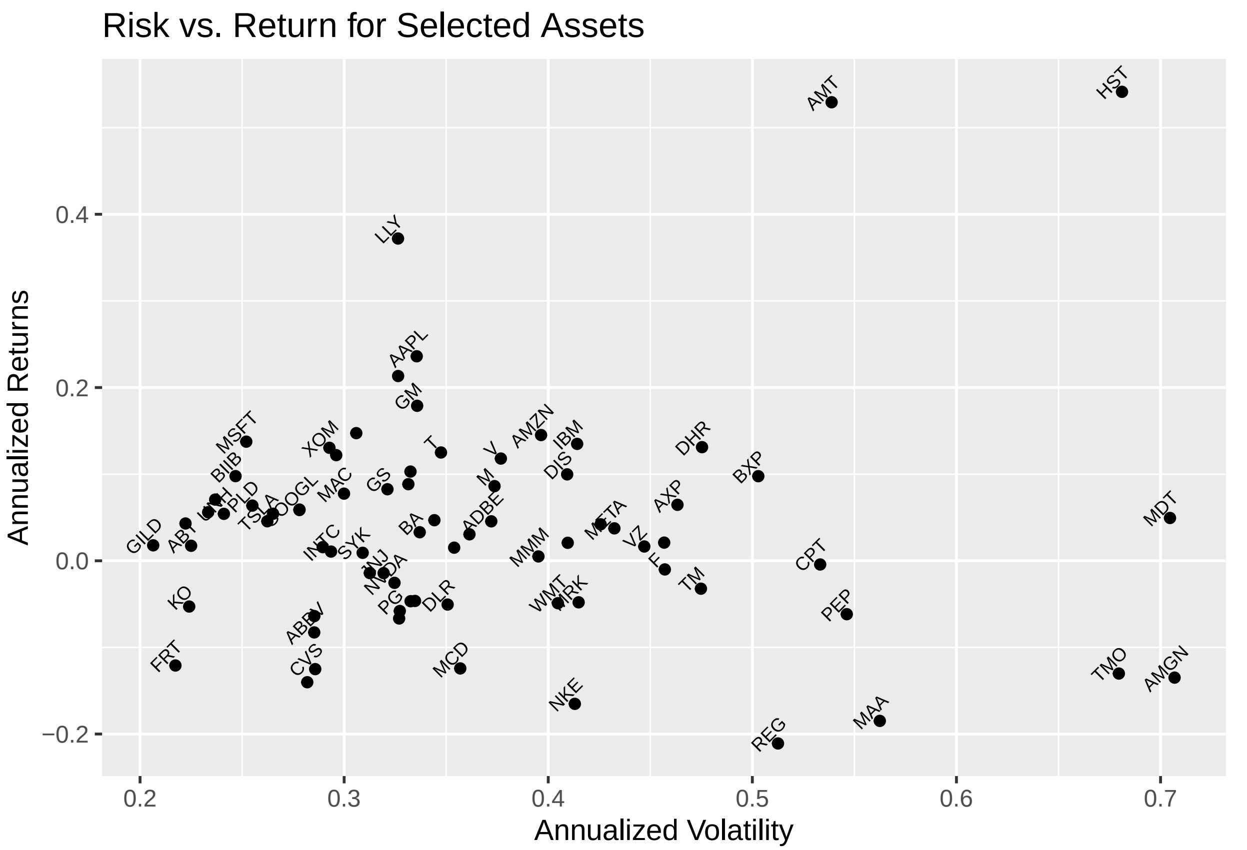
- In this section, we fetched historical stock prices from 2020 to 2022 for a selection of 76 assets and compute their log returns, which are crucial for assessing past performance and volatility.
- We ensured data integrity by omitting NAs and then update our asset list to only those with complete data.
- Lastly, we visualized the risk versus return of these assets using a scatter plot with clear labels for each asset, aiding in their comparison and selection for portfolio optimization.

```
symbols <- c("AAPL", "MSFT", "GOOGL", "AMZN", "META", "TSLA", "JNJ", "V", "PG",
             "NVDA", "DIS", "PEP", "TM", "KO", "NKE", "ADBE", "NFLX", "INTC", "CSCO",
             "XOM", "MCD", "BA", "MMM", "GS", "DOW", "JPM", "AXP", "WMT", "IBM",
             "GE", "F", "GM", "T", "VZ", "PFE", "MRK", "GILD", "BMY", "CNC",
             "ABT", "AMGN", "LLY", "MDT", "SYK", "TMO", "BIIB", "ABBV", "DHR",
             "CVS", "UNH", "O", "BXP", "SPG", "AMT", "DLR", "EQIX", "WY", "AVB",
             "EQR", "ESS", "MAA", "CPT", "UDR", "AIV", "ARE", "PLD", "VNO", "HST",
             "SLG", "KIM", "MAC", "REG", "FRT", "TGT", "KSS", "M")

# Fetching historical data for each symbol and calculate log returns
data <- new.env()
getSymbols(symbols, src = 'yahoo', from = '2020-01-01', to = '2023-12-31', env = data, auto.assign = TRU
```

```
##  [1] "AAPL"  "MSFT"  "GOOGL" "AMZN"  "META"  "TSLA"  "JNJ"   "V"     "PG"
## [10] "NVDA"  "DIS"   "PEP"   "TM"    "KO"    "NKE"   "ADBE"  "NFLX"  "INTC"
## [19] "CSCO"  "XOM"   "MCD"   "BA"    "MMM"   "GS"    "DOW"   "JPM"   "AXP"
## [28] "WMT"   "IBM"   "GE"    "F"     "GM"    "T"     "VZ"    "PFE"   "MRK"
## [37] "GILD"  "BMY"   "CNC"   "ABT"   "AMGN"  "LLY"   "MDT"   "SYK"   "TMO"
## [46] "BIIB"  "ABBV"  "DHR"   "CVS"   "UNH"   "O"     "BXP"   "SPG"   "AMT"
## [55] "DLR"   "EQIX"  "WY"    "AVB"   "EQR"   "ESS"   "MAA"   "CPT"   "UDR"
## [64] "AIV"   "ARE"   "PLD"   "VNO"   "HST"   "SLG"   "KIM"   "MAC"   "REG"
## [73] "FRT"   "TGT"   "KSS"   "M"
```

```
log_returns <- lapply(ls(data), function(symbol) {
  prices <- get(symbol, envir = data)
  Return.calculate(Cl(prices), method="log")
})
# Assuming you have already loaded the necessary libraries and fetched the data

# Preprocess log returns: remove NA, ensure equal length
log_returns <- lapply(log_returns, na.omit)
equal_length <- min(sapply(log_returns, length))
log_returns <- lapply(log_returns, function(x) x[1:equal_length])

# Now we can safely bind the log returns into a matrix
log_returns_matrix <- do.call(cbind, log_returns)
names(log_returns) <- symbols

# Assuming you have calculated the annualized returns and volatilities
annualized_returns <- sapply(log_returns, function(x) mean(x) * 252)
annualized_volatility <- sapply(log_returns, function(x) sd(x) * sqrt(252))

# Now create plot_data dataframe
plot_data <- data.frame(
  Symbol = names(log_returns),
  Returns = annualized_returns,
```

```
  Volatility = annualized_volatility
)

# Proceed with ggplot
ggplot(plot_data, aes(x = Volatility, y = Returns, label = Symbol)) +
  geom_point() +
  geom_text(check_overlap = TRUE, vjust = -0.5, hjust = 0.5, size = 2.5, angle = 45) +
  xlab("Annualized Volatility") + ylab("Annualized Returns") +
  ggtitle("Risk vs. Return for Selected Assets")
```



The scatter plot illustrates the relationship between risk and return for various financial assets, depicting varying levels of potential return for their associated risks. - Assets like 'HST' and 'AMT' appear to offer higher returns but also come with higher volatility, suggesting a higher risk. - In contrast, assets such as 'KO' and 'PG' show lower volatility, implying they are less risky, though they may offer more modest returns.

Part 2(b): Asset Selection Using GA

- We defined a GA to select an optimal subset of assets from our pool, aiming to construct a portfolio with a predefined number of assets (preferred_asset_count).
- The fitness_function_selection is crafted to compute the Sharpe ratio, a measure of risk-adjusted return, of a portfolio composed of selected assets based on their log returns.
- We introduce a penalty to discourage deviation from the preferred number of assets, ensuring the portfolio doesn't deviate too much from the desired complexity.
- The genetic algorithm (ga_selection) searches for the best combination of assets that maximizes this fitness function. After running the GA, we identified the indices and symbols of the chosen assets and display them.

```r
preferred_asset_count <- 10
penalty_factor <- 1000

fitness_function_selection <- function(subset_indices) {
  selected_returns <- do.call(cbind, log_returns)[, subset_indices == 1, drop = FALSE]

  if(ncol(selected_returns) == 0) return(-Inf)

  penalty <- abs(preferred_asset_count - sum(subset_indices)) * penalty_factor
  portfolio_returns <- rowMeans(selected_returns)
  sharpe_ratio <- mean(portfolio_returns) / sd(portfolio_returns)

  if(is.nan(sharpe_ratio) || is.infinite(sharpe_ratio)) return(-Inf)

  return(sharpe_ratio - penalty)
}
```

GA Selection

- The population size (popSize) is set to 50 to ensure a good diversity in solutions while keeping computation times reasonable.
- Maximum iterations (maxiter) is capped at 100 to balance between allowing enough generations for convergence and computational efficiency.

```r
ga_selection <- ga(type = "binary", fitness = fitness_function_selection,
                   nBits = length(symbols), popSize = 50, maxiter = 100)

selected_indices <- which(ga_selection@solution == 1)
selected_symbols <- symbols[selected_indices]
cat("Selected assets:", paste(selected_symbols, collapse = ", "), "\n")
```

```
## Selected assets: AAPL, TSLA, INTC, XOM, F, GM, EQR, CPT, VNO, HST
```

Part 2(c): Portfolio Optimization with Selected Assets

- The code optimizes a financial portfolio by finding the best weights for selected assets to maximize the Sharpe ratio.
- It ensures that the portfolio's risk-adjusted return is as high as possible, penalizing any deviation from the desired number of assets

```r
log_returns_matrix <- do.call(cbind, log_returns[selected_symbols])

fitness_function <- function(weights) {
  weights <- weights / sum(weights)
  portfolio_returns <- log_returns_matrix %*% matrix(weights, ncol = 1)

  if(any(is.na(portfolio_returns))) return(-Inf)

  portfolio_sd <- sd(portfolio_returns)
  if(portfolio_sd == 0) return(-Inf)

  sharpe_ratio <- mean(portfolio_returns) / portfolio_sd
  return(sharpe_ratio)
}
```

```
ga_result <- ga(type = "real-valued", fitness = fitness_function,
                lower = rep(0, length(selected_symbols)), upper = rep(1, length(selected_symbols)),
                popSize = 50, maxiter = 100, run = 50)

best_weights <- ga_result@solution
best_weights <- best_weights / sum(best_weights)
```

Part 2(d): Visualization of Portfolio Performance

```
portfolio_log_returns <- log_returns_matrix %*% matrix(best_weights, ncol = 1)
portfolio_simple_returns <- exp(portfolio_log_returns) - 1
cumulative_returns <- cumprod(1 + portfolio_simple_returns) - 1

# Convert trading days to years for the x-axis
time_in_years <- seq_along(cumulative_returns) / 252

# Plot with years on the x-axis
plot(time_in_years, cumulative_returns, type = 'l', col = 'red',
     main = "Optimized Portfolio Performance",
     xlab = "Time (Years)", ylab = "Cumulative Returns")
```