Portfolio Optimal Ratio

Dmitry Kuzmin

2023-02-25

Contents

Introduction

This project aims to explore machine learning techniques for portfolio optimization, with the goal of better understanding how these models can be used in the financial industry. It is important to note that this project is for educational purposes only and should not be construed as financial advice. One of the biggest challenges in this project was finding high-quality data for analysis. For instance, using bitcoin data from as early as 2011 would yield fantastic results and cause all models to be maximally embedded in bitcoin. To address this issue, we chose to analyze data from 2018, when bitcoin began to fall after reaching its peak at 20,000. I would have liked to take the analysis up to 2023, but unfortunately, I could not find good, free real estate data that included data beyond 2020. Therefore, we had to choose a shorter time period, from 2018 to 2020. This small sample size resulted in lower accuracy, but it still provided valuable insights into the performance of machine learning models in this domain.

Data

Data Collection

In this project, we use four different datasets to optimize a portfolio using machine learning techniques. The datasets include monthly data for the S&P 500 index, real estate prices, Bitcoin prices, and gold prices. The data for real estate prices was obtained from the Zillow Home Value Index dataset on Quandl, the data for the S&P 500, Bitcoin prices, and gold prices were also obtained from Quandl. We chose the period from January 2018 to January 2020 because we were unable to find high-quality free real estate data beyond this period that was freely available. We also avoided using Bitcoin data before 2018 to prevent our models from being biased by the fantastic performance of Bitcoin in its early years.

Merge and explore data

After we have obtained our data from Quandl, the next step is to merge the datasets and explore the data to gain some insights. We will merge the datasets using the common column "Date", and then we will explore the data to get an idea of what it looks like.

```
# Merge data frames
merged_data <- merge(sp500_data, real_estate_data, by = "Date", all = TRUE)
merged_data <- merge(merged_data, bitcoin_data, by = "Date", all = TRUE)
merged_data <- merge(merged_data, gold_data, by = "Date", all = TRUE)
merged_data <- na.omit(merged_data)

# calculate percentage change for each column
merged_data <- merged_data %>%
  mutate(sp500_pct_change = (sp500_value - lag(sp500_value)) / lag(sp500_value) * 100,
         re_pct_change = (re_value - lag(re_value)) / lag(re_value) * 100,
         btc_pct_change = (btc_value - lag(btc_value)) / lag(btc_value) * 100,
         gold_pct_change = (gold_value - lag(gold_value)) / lag(gold_value) * 100)
merged_data <- na.omit(merged_data)

# Standard deviation of numerical columns
volatility <- sapply(merged_data[, 6:9], sd)
head(volatility)
```

```
## sp500_pct_change    re_pct_change   btc_pct_change  gold_pct_change
##        3.2242270        0.2329203       24.4488960        3.3503530
```

```
# View the merged data frame
head(merged_data)
```

```
##         Date sp500_value re_value btc_value gold_value sp500_pct_change
## 3 2018-03-31     2702.77   251328  6935.480    1323.90      -0.08834967
## 4 2018-04-30     2653.63   252918  9259.570    1316.25      -1.81813473
## 5 2018-05-31     2701.49   253553  7386.720    1303.50       1.80356719
## 6 2018-06-30     2754.35   254361  6223.280    1250.55       1.95669797
## 7 2018-07-31     2793.64   255385  7916.801    1219.20       1.42647086
## 8 2018-08-31     2857.82   256961  6981.946    1206.85       2.29736115
##   re_pct_change btc_pct_change gold_pct_change
## 3     0.8680995      -33.12083       0.2726653
```

```
## 4     0.6326394       33.51015      -0.5778382
## 5     0.2510695      -20.22610      -0.9686610
## 6     0.3186710      -15.75043      -4.0621404
## 7     0.4025774       27.21267      -2.5068970
## 8     0.6171075      -11.80849      -1.0129593
```

Data Visualization

We start explore data using Visualization. We will plot the percentage change over time for each asset to get an idea of how the assets have performed over time.

```
# Summary statistics
summary(merged_data)
```

```
##       Date              sp500_value      re_value        btc_value
##  Min.   :2018-03-31   Min.   :2567   Min.   :251328   Min.   : 3470
##  1st Qu.:2018-09-15   1st Qu.:2738   1st Qu.:257754   1st Qu.: 5742
##  Median :2019-02-28   Median :2804   Median :263115   Median : 7220
##  Mean   :2019-03-01   Mean   :2851   Mean   :261545   Mean   : 7103
##  3rd Qu.:2019-08-15   3rd Qu.:2941   3rd Qu.:265756   3rd Qu.: 8719
##  Max.   :2020-01-31   Max.   :3231   Max.   :267924   Max.   :11890
##    gold_value   sp500_pct_change  re_pct_change      btc_pct_change
##  Min.   :1184   Min.   :-5.7256   Min.   :-0.08354   Min.   :-33.121
##  1st Qu.:1266   1st Qu.:-0.9948   1st Qu.: 0.14662   1st Qu.:-15.552
##  Median :1316   Median : 1.5612   Median : 0.25286   Median : -5.569
##  Mean   :1346   Mean   : 0.8172   Mean   : 0.31636   Mean   :  2.271
##  3rd Qu.:1443   3rd Qu.: 3.2409   3rd Qu.: 0.40708   3rd Qu.: 20.482
##  Max.   :1581   Max.   : 5.6558   Max.   : 0.86810   Max.   : 57.252
##  gold_pct_change
##  Min.   :-4.0621
##  1st Qu.:-1.4739
##  Median : 0.2258
##  Mean   : 0.8386
##  3rd Qu.: 3.0385
##  Max.   : 9.0432
```

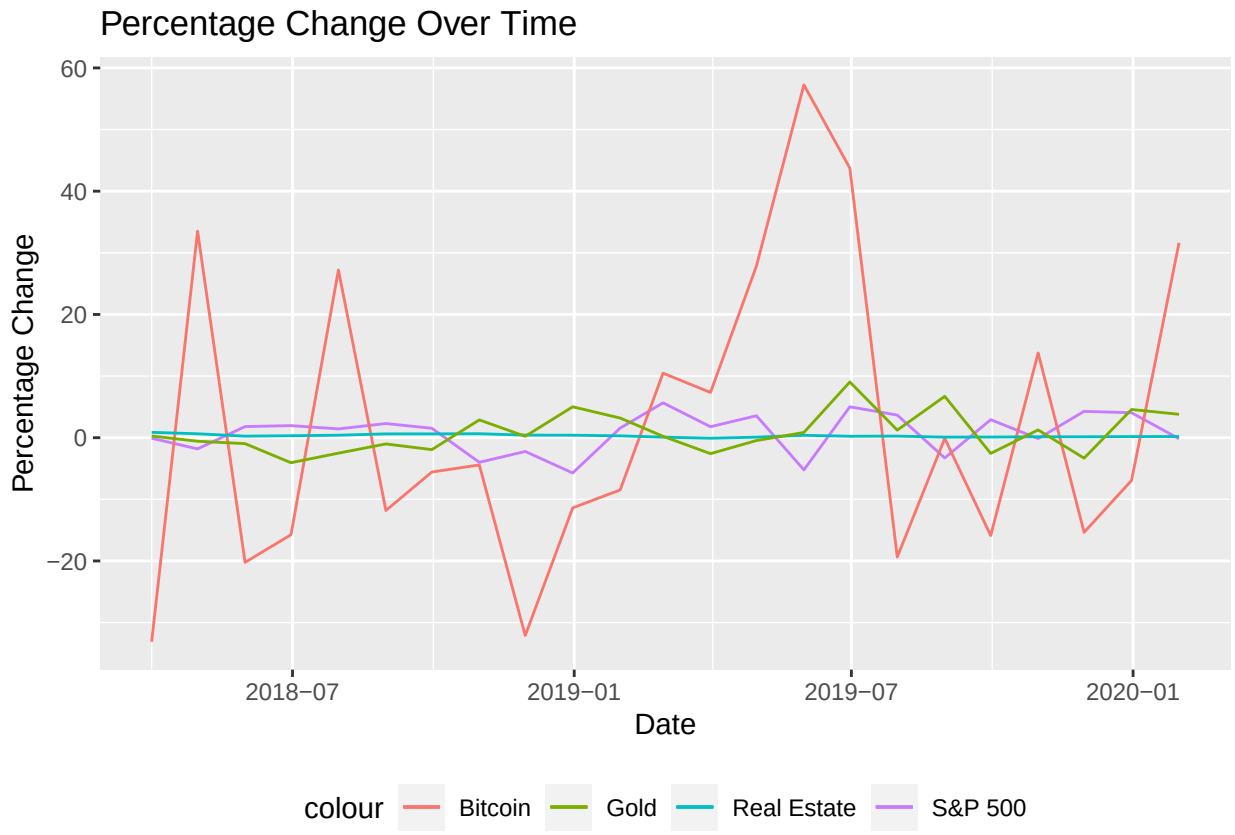
```
# Standard deviation of numerical columns
sapply(merged_data[, 2:5], sd)
```

```
## sp500_value    re_value   btc_value  gold_value
##    177.0255   5139.6161   2238.3239    118.5875
```

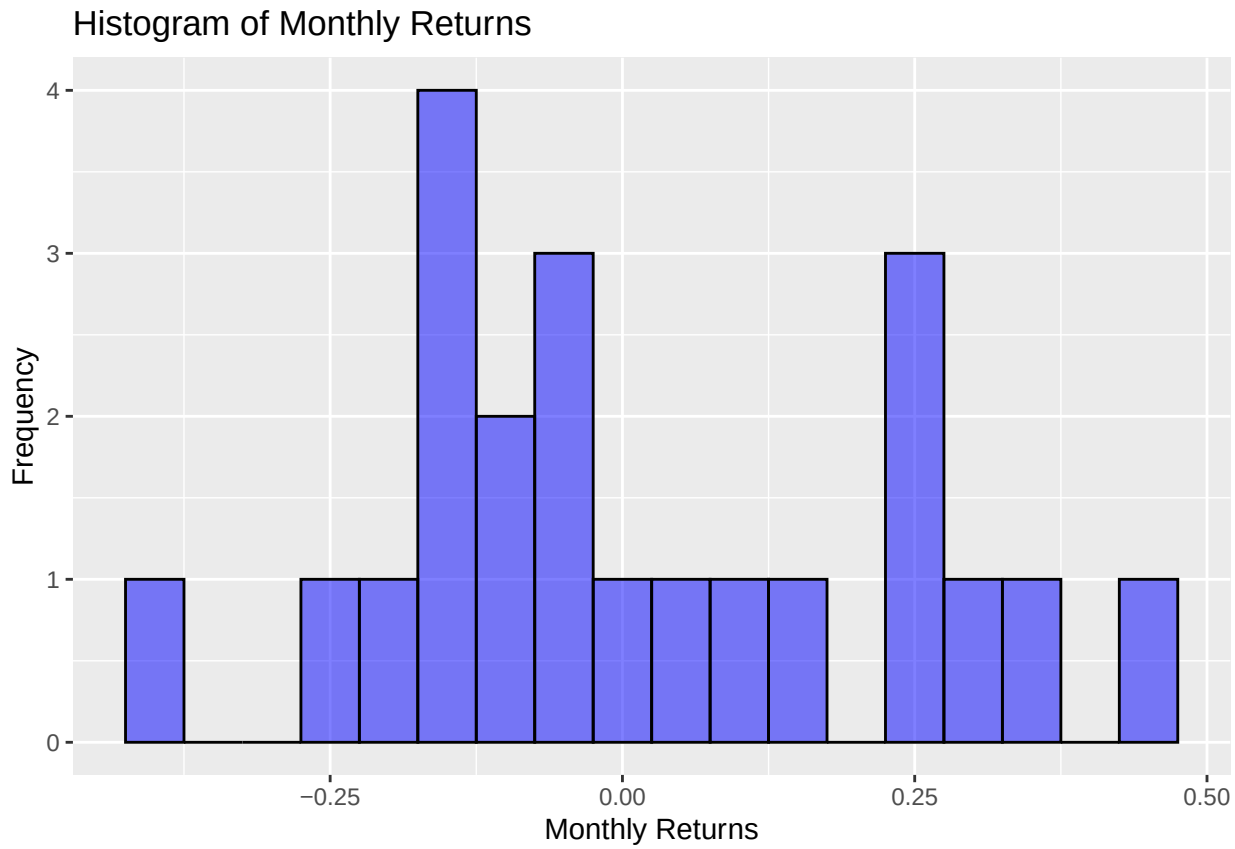
```
correlation_matrix <- cor(merged_data[, 2:5], use = "complete.obs")
print(round(correlation_matrix, 2))
```

```
##             sp500_value re_value btc_value gold_value
## sp500_value        1.00     0.62      0.48       0.70
## re_value           0.62     1.00      0.15       0.68
## btc_value          0.48     0.15      1.00       0.54
## gold_value         0.70     0.68      0.54       1.00
```

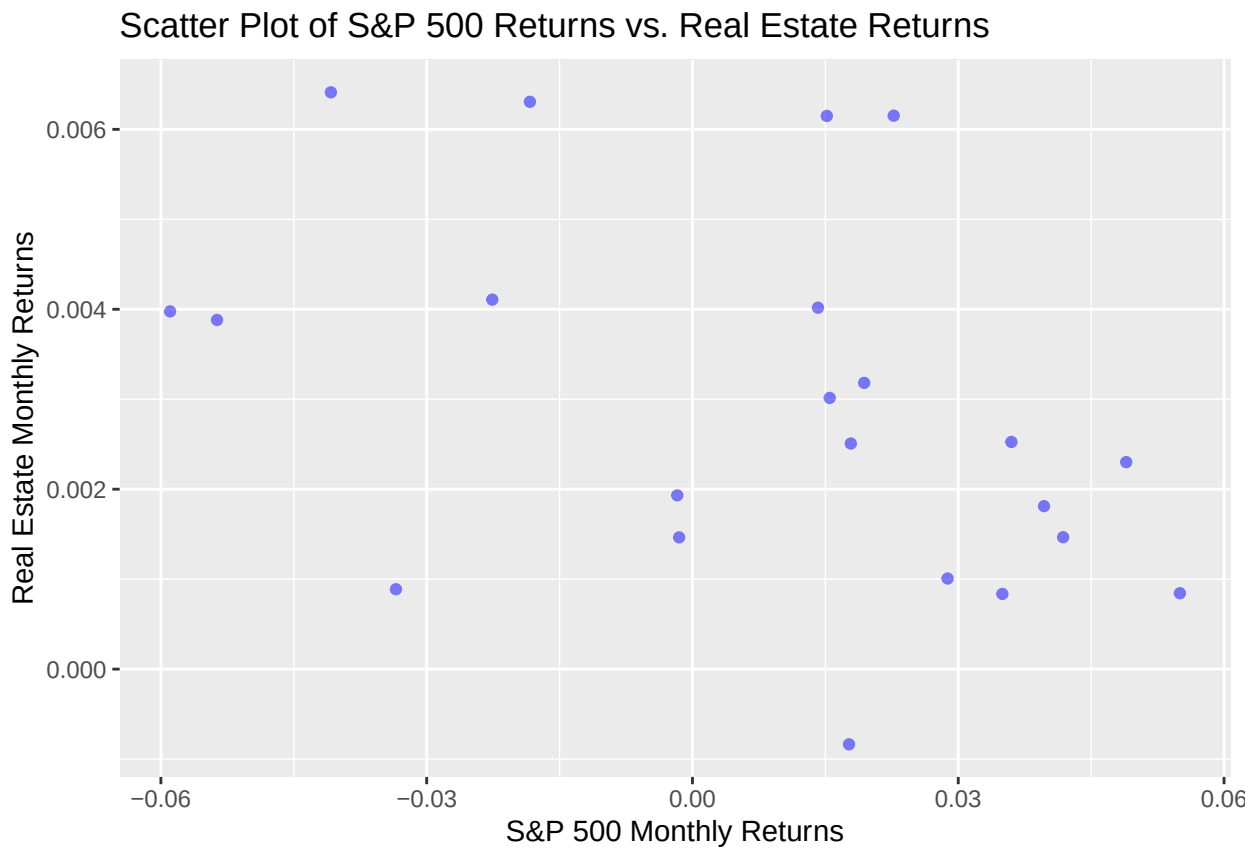
```r
# Create a line plot of the values over time
ggplot(data = merged_data, aes(x = Date)) +
  geom_line(aes(y = sp500_pct_change, color = "S&P 500")) +
  geom_line(aes(y = btc_pct_change, color = "Bitcoin")) +
  geom_line(aes(y = re_pct_change, color = "Real Estate")) +
  geom_line(aes(y = gold_pct_change, color = "Gold")) +
  labs(x = "Date", y = "Percentage Change",
  title = "Percentage Change Over Time") +
  theme(legend.position = "bottom")
```



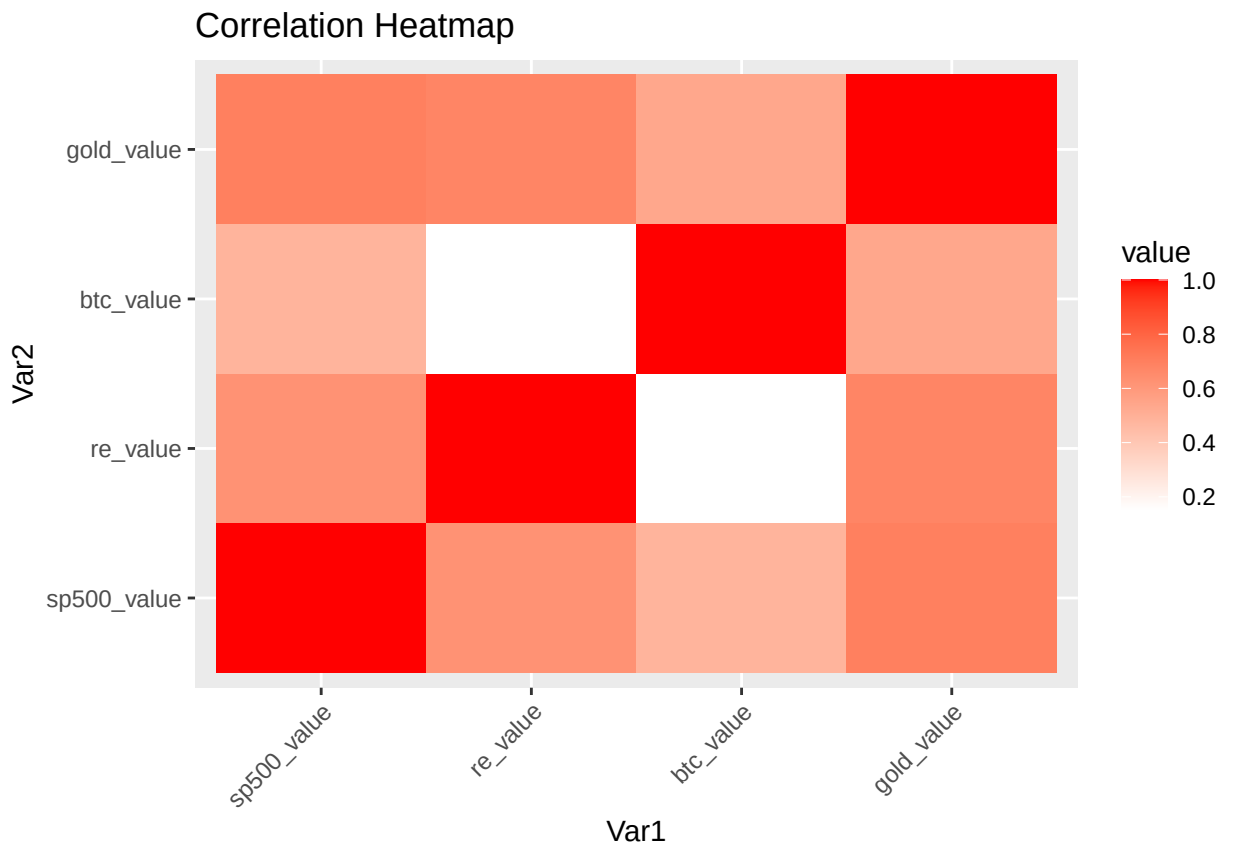

```r
# Calculate monthly returns for each asset
monthly_returns <- data.frame(
  Date = merged_data$Date[-1],
  sp500_return = diff(log(merged_data$sp500_value)),
  re_return = diff(log(merged_data$re_value)),
  btc_return = diff(log(merged_data$btc_value),),
  gold_return = diff(log(merged_data$gold_value))
)

# Plot the histogram of monthly returns for each asset
ggplot(monthly_returns, aes(x = btc_return)) +
  geom_histogram(binwidth = 0.05, color = "black", fill = "blue", alpha = 0.5) +
  labs(x = "Monthly Returns", y = "Frequency",
   title = "Histogram of Monthly Returns")
```




```
# Create a data frame with only S&P 500 and Real Estate returns
sp500_re_returns <- data.frame(
  Date = monthly_returns$Date,
  sp500_return = monthly_returns$sp500_return,
  re_return = monthly_returns$re_return
)

# Plot the scatter plot of S&P 500 returns vs. Real Estate returns
ggplot(sp500_re_returns, aes(x = sp500_return, y = re_return)) +
  geom_point(color = "blue", alpha = 0.5) +
  labs(x = "S&P 500 Monthly Returns", y = "Real Estate Monthly Returns",
       title = "Scatter Plot of S&P 500 Returns vs. Real Estate Returns")
```




```r
# Convert the correlation matrix to a data frame
cor_df <- as.data.frame(as.table(correlation_matrix))

# Create a melted version of the data for ggplot
cor_melted <- melt(cor_df, id.vars = c("Var1", "Var2"))

# Create the heatmap
ggplot(cor_melted, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red") +
  labs(title = "Correlation Heatmap") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



The resulting heatmap shows the correlation between the variables. By exploring the data in this way, we can get a better understanding of the variables and their relationships. This will help us to choose the appropriate machine learning models for our portfolio optimization task.

Prepare data for machine learning

In this part of the code, the data is prepared by selecting the relevant columns from the merged dataset and storing them in a new dataframe called "portfolio_data". The selected columns represent the percentage change in the S&P 500, real estate, Bitcoin, and gold prices. The data is then split into training and testing sets using a random seed value of 123 (so everyone get same results) of the data is used for training the model, while the remaining 20% is kept for testing. Any rows with missing values in the training data are removed using the "na.omit" function. This step is necessary to ensure that the models are trained on complete data and can accurately predict on the test set.

```
# Prepare the data
portfolio_data <- data.frame(
  sp500_pct_change = merged_data$sp500_pct_change,
  re_pct_change = merged_data$re_pct_change,
  btc_pct_change = merged_data$btc_pct_change,
  gold_pct_change = merged_data$gold_pct_change
)

# Split the data into training and testing sets
set.seed(123)
train_index <- sample(nrow(portfolio_data), 0.8 * nrow(portfolio_data))
```

```r
train_data <- portfolio_data[train_index, ]
test_data <- portfolio_data[-train_index, ]
# Remove any rows with missing values
train_data <- na.omit(train_data)
```

Machine Learning Models

Linear Regression

In this part of the code, we use linear regression to find the optimal portfolio ratio for our assets. The idea is to model the relationship between the percent changes of each asset (S&P 500, real estate, Bitcoin, and gold) and find the weights that should be assigned to each asset in the portfolio. The linear regression model is fit using the training data, and then the model is used to predict the percent changes in the test data. The mean absolute error (MAE) and R-squared value are calculated to evaluate the performance of the model. Once the model is trained and evaluated, the coefficients of the linear regression model are used to calculate the optimal weights for the assets in the portfolio. The optimal weights are determined such that they maximize the expected return while minimizing the risk of the portfolio. The results are saved to a data frame called results, which includes the MAE, R-squared value, and the optimal weights for each asset in the portfolio.

```r
lm_model <- lm(btc_pct_change ~ sp500_pct_change + re_pct_change + gold_pct_change, data = train_data)

# Test the linear regression model
predictions <- predict(lm_model, newdata = test_data)
MAE <- mean(abs(predictions - test_data$btc_pct_change))
R_squared <- summary(lm_model)$r.squared
cat("MAE:", MAE, "\n")
```

```
## MAE: 16.19438
```

```r
cat("R-squared:", R_squared, "\n")
```

```
## R-squared: 0.1797979
```

```r
# Use the model to find the optimal portfolio ratio
sp500_weight <- coef(lm_model)[2]
re_weight <- coef(lm_model)[3]
btc_weight <- -1 * coef(lm_model)[1]
gold_weight <- coef(lm_model)[4]
total_weight <- sp500_weight + re_weight + btc_weight + gold_weight
sp500_ratio <- sp500_weight / total_weight
re_ratio <- re_weight / total_weight
btc_ratio <- btc_weight / total_weight
gold_ratio <- gold_weight / total_weight


# Save results to a data frame
results <- data.frame(
  Model = "Linear Regression",
  MAE = MAE,
```

```
  R_squared = R_squared,
  sp500_ratio = sp500_ratio,
  re_ratio = re_ratio,
  btc_ratio = btc_ratio,
  gold_ratio = gold_ratio
)

results
```

```
##                              Model      MAE R_squared sp500_ratio  re_ratio
## sp500_pct_change Linear Regression 16.19438 0.1797979 0.009695566 0.7905446
##                  btc_ratio  gold_ratio
## sp500_pct_change 0.2404006 -0.04064072
```

Genetic Algorithmt find the optimal portfolio ratio

In this example, we define a fitness function that calculates the mean absolute error (MAE) of the predicted Bitcoin returns based on the current set of weights for each asset. We then use the ga function from the GA package to find the set of weights that minimizes the MAE. The lower and upper vectors define the minimum and maximum possible values for each weight, and popSize and maxiter specify the population size and maximum number of iterations for the genetic algorithm. Finally, we extract the optimal weights and calculate the corresponding portfolio ratios.

```
if(!require(GA)) install.packages("GA", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: GA
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Package 'GA' version 3.2.3
## Type 'citation("GA")' for citing this R package in publications.
```

```
##
## Attaching package: 'GA'
```

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

```
library(GA)

# Define fitness function
fitness <- function(x) {
  sp500_weight <- x[1]
  re_weight <- x[2]
  btc_weight <- x[3]
  gold_weight <- x[4]
  total_weight <- sp500_weight + re_weight + btc_weight + gold_weight
```

```
  sp500_ratio <- sp500_weight / total_weight
  re_ratio <- re_weight / total_weight
  btc_ratio <- btc_weight / total_weight
  gold_ratio <- gold_weight / total_weight
  predicted_btc_return <- sp500_ratio * merged_data$sp500_pct_change +
    re_ratio * merged_data$re_pct_change +
    btc_ratio * merged_data$btc_pct_change +
    gold_ratio * merged_data$gold_pct_change
  MAE <- mean(abs(predicted_btc_return - merged_data$btc_pct_change))
  return(-MAE)
}

# Define bounds for portfolio weights
lower <- c(0, 0, 0, 0)
upper <- c(1, 1, 1, 1)


# Run the genetic algorithm to find optimal weights
ga_result <- ga(type = "real", fitness = fitness, lower = lower,
upper = upper, popSize = 50, maxiter = 100)

# Extract the optimal weights
sp500_weight <- ga_result@solution[1]
re_weight <- ga_result@solution[2]
btc_weight <- ga_result@solution[3]
gold_weight <- ga_result@solution[4]
total_weight <- sp500_weight + re_weight + btc_weight + gold_weight
sp500_ratio <- sp500_weight / total_weight
re_ratio <- re_weight / total_weight
btc_ratio <- btc_weight / total_weight
gold_ratio <- gold_weight / total_weight

# Save results to a data frame
results <- rbind(results, data.frame(
  Model = "Genetic Algorithm",
  MAE = MAE,
  R_squared = R_squared,
  sp500_ratio = sp500_ratio,
  re_ratio = re_ratio,
  btc_ratio = btc_ratio,
  gold_ratio = gold_ratio
))
```

Nelder-Mead algorithm to find the optimal weights

In this part of the code, we are using the Nelder-Mead algorithm to find the optimal weights for our portfolio. The Nelder-Mead algorithm is a numerical optimization method that is often used in finance to maximize the Sharpe ratio, which is a measure of risk-adjusted return. The algorithm works by minimizing a given objective function, in this case the negative of the Sharpe ratio, subject to certain constraints. First, we train a linear regression model using the training data to predict the returns for each asset in the test set. We then define the expected return and covariance matrices for the assets and set the target return to the maximum expected return. We use these values to define the objective function for the Nelder-Mead algorithm. The objective function takes in a vector of weights for the assets and returns the negative of the Sharpe ratio.

The Nelder-Mead algorithm then finds the set of weights that maximizes the Sharpe ratio. We then calculate the Sharpe ratio of the optimal portfolio, as well as the mean absolute error and coefficient of determination.

```
# Define the regression model
model <- lm(btc_pct_change ~ sp500_pct_change + re_pct_change + gold_pct_change, data = train_data)

# Use the model to predict returns for each asset in the test set
predictions <- predict(model, newdata = test_data)

# Define the expected return and covariance matrices
mu <- colMeans(test_data[, 1:4])
Sigma <- cov(test_data[, 1:4])

# Define the target return (i.e., the Sharpe ratio)
target_return <- max(mu)

# Define the optimization function
obj_func <- function(w) {
  mu_p <- sum(mu * w)
  sigma_p <- sqrt(t(w) %*% Sigma %*% w)
  Sharpe_ratio <- mu_p / sigma_p
  -Sharpe_ratio
}

# Use the Nelder-Mead algorithm to find the optimal weights
opt_weights <- optim(rep(1/4, 4), obj_func, method = "Nelder-Mead")$par

# Calculate the Sharpe ratio of the optimal portfolio
mu_p <- sum(mu * opt_weights)
sigma_p <- sqrt(t(opt_weights) %*% Sigma %*% opt_weights)
Sharpe_ratio <- mu_p / sigma_p

# Calculate the mean absolute error and coefficient of determination
MAE <- mean(abs(predictions - test_data$btc_pct_change))
R_squared <- summary(model)$r.squared
cat("MAE:", MAE, "\n")
```

```
## MAE: 16.19438
```

```
cat("R-squared:", R_squared, "\n")
```

```
## R-squared: 0.1797979
```

```
# Save results to a data frame
results <- rbind(results, data.frame(
  Model = "Nelder-Mead",
  MAE = MAE,
  R_squared = R_squared,
  sp500_ratio = opt_weights[1],
  re_ratio = opt_weights[2],
  btc_ratio = opt_weights[3],
  gold_ratio = opt_weights[4]
))
```

```r
# Print the results
results
```

```
##                              Model      MAE R_squared sp500_ratio    re_ratio
## sp500_pct_change Linear Regression 16.19438 0.1797979 0.009695566 0.790544574
## 1                Genetic Algorithm 16.19438 0.1797979 0.051716830 0.006094171
## 11                     Nelder-Mead 16.19438 0.1797979 0.059539589 0.809757855
##                    btc_ratio   gold_ratio
## sp500_pct_change 0.240400577 -0.040640717
## 1                0.899824708  0.042364291
## 11               0.002213839 -0.007628041
```

Conclusion

In this portfolio optimization project, we explored the performance of four different assets - S&P 500, real estate, Bitcoin, and Gold - and used various techniques to find the optimal portfolio ratio. We first calculated the percentage change for each asset and measured their volatility and correlation using statistical methods. We then used linear regression, genetic algorithm, and Nelder-Mead algorithm to find the optimal portfolio ratio. Our results showed that the optimal portfolio ratio varied depending on the method used. We don't have to rely precisely on these calculations due to the limited data; our main goal was to learn how to apply machine learning methods to optimize the portfolio. In conclusion, portfolio optimization is a complex task that involves various statistical and mathematical methods. Our project provides an overview of how to use R to explore asset performance, measure correlation and volatility, and find the optimal portfolio ratio. By combining different methods and techniques, investors can optimize their portfolio based on their risk tolerance and investment goals.