FE 513 HW 1 SHUBHAM

Shubham Narkhede- 20011019

2023-09-30

PART 1

Part 1.1

```r
# Create two vectors with 10 random numbers each
vector1 <- runif(10)
print(vector1)
```

```
##  [1] 0.8334263 0.8260904 0.9913024 0.2927196 0.4399142 0.4131399 0.2224776
##  [8] 0.9887734 0.6268252 0.9041487
```

```r
vector2 <- runif(10)
print(vector2)
```

```
##  [1] 0.3210609 0.8284027 0.3160880 0.2344815 0.6258761 0.6068147 0.2150288
##  [8] 0.4593601 0.3024314 0.8668849
```

```r
# Append the second vector to the first one
combined_vector <- c(vector1, vector2)
combined_vector
```

```
##  [1] 0.8334263 0.8260904 0.9913024 0.2927196 0.4399142 0.4131399 0.2224776
##  [8] 0.9887734 0.6268252 0.9041487 0.3210609 0.8284027 0.3160880 0.2344815
## [15] 0.6258761 0.6068147 0.2150288 0.4593601 0.3024314 0.8668849
```

```r
# Calculate the mean of the combined vector
mean_combined <- mean(combined_vector)
mean_combined
```

```
## [1] 0.5657623
```

```r
# Print 'True' if a number is greater than the mean, else 'False'
result <- ifelse(combined_vector > mean_combined, 'True', 'False')
print(result)
```

```
##  [1] "True"  "True"  "True"  "False" "False" "False" "False" "True"  "True"
## [10] "True"  "False" "True"  "False" "False" "True"  "True"  "False" "False"
## [19] "False" "True"
```

##PART 1.2

```r
# Create a vector with 100 random numbers
vector_matrix <- runif(100)
vector_matrix
```

```
##   [1] 0.16842028 0.75606629 0.14280543 0.21712503 0.40219821 0.93504847
##   [7] 0.77691842 0.31712134 0.44223778 0.98095037 0.91022815 0.72989593
##  [13] 0.47581982 0.75077768 0.42602163 0.58950706 0.57817828 0.71711009
##  [19] 0.03980728 0.06987698 0.28405701 0.87666870 0.25142080 0.84038227
##  [25] 0.32023494 0.46624324 0.39069637 0.65199525 0.17223613 0.55352070
##  [31] 0.43710677 0.07779830 0.79173206 0.60220024 0.83341888 0.13309247
##  [37] 0.63489682 0.30535822 0.32631368 0.90106745 0.62084759 0.31163419
##  [43] 0.87540704 0.23475172 0.58182321 0.69899255 0.85560577 0.81356429
##  [49] 0.02807054 0.53495305 0.89752908 0.43762604 0.80377302 0.34314805
##  [55] 0.80959902 0.50670199 0.73516268 0.78202573 0.77271202 0.93244927
##  [61] 0.63025028 0.31958476 0.92655021 0.11683922 0.78990728 0.84558798
##  [67] 0.92846577 0.66987155 0.95857661 0.25596077 0.82670839 0.65535306
##  [73] 0.91255735 0.01083941 0.19011180 0.40048211 0.77406804 0.69383729
##  [79] 0.75480396 0.37906312 0.22345926 0.50415756 0.29098682 0.72302158
##  [85] 0.44537078 0.52990017 0.91746363 0.47511266 0.67288952 0.12283416
##  [91] 0.10121895 0.96308071 0.56488617 0.18525646 0.31412970 0.56626043
##  [97] 0.74116643 0.94542656 0.27697711 0.27005991
```

```r
# Transfer the vector into a 10 by 10 matrix M
M <- matrix(vector_matrix, nrow = 10, ncol = 10)
M
```

```
##            [,1]       [,2]      [,3]      [,4]       [,5]      [,6]      [,7]
##  [1,] 0.1684203 0.91022815 0.2840570 0.4371068 0.62084759 0.8975291 0.6302503
##  [2,] 0.7560663 0.72989593 0.8766687 0.0777983 0.31163419 0.4376260 0.3195848
##  [3,] 0.1428054 0.47581982 0.2514208 0.7917321 0.87540704 0.8037730 0.9265502
##  [4,] 0.2171250 0.75077768 0.8403823 0.6022002 0.23475172 0.3431480 0.1168392
##  [5,] 0.4021982 0.42602163 0.3202349 0.8334189 0.58182321 0.8095990 0.7899073
##  [6,] 0.9350485 0.58950706 0.4662432 0.1330925 0.69899255 0.5067020 0.8455880
##  [7,] 0.7769184 0.57817828 0.3906964 0.6348968 0.85560577 0.7351627 0.9284658
##  [8,] 0.3171213 0.71711009 0.6519952 0.3053582 0.81356429 0.7820257 0.6698715
##  [9,] 0.4422378 0.03980728 0.1722361 0.3263137 0.02807054 0.7727120 0.9585766
## [10,] 0.9809504 0.06987698 0.5535207 0.9010675 0.53495305 0.9324493 0.2559608
##             [,8]      [,9]     [,10]
##  [1,] 0.82670839 0.2234593 0.1012189
##  [2,] 0.65535306 0.5041576 0.9630807
##  [3,] 0.91255735 0.2909868 0.5648862
##  [4,] 0.01083941 0.7230216 0.1852565
##  [5,] 0.19011180 0.4453708 0.3141297
##  [6,] 0.40048211 0.5299002 0.5662604
##  [7,] 0.77406804 0.9174636 0.7411664
##  [8,] 0.69383729 0.4751127 0.9454266
##  [9,] 0.75480396 0.6728895 0.2769771
## [10,] 0.37906312 0.1228342 0.2700599
```

```r
# Find the transpose of matrix M
MT <- t(M)
MT
```

```
##              [,1]      [,2]      [,3]       [,4]      [,5]      [,6]      [,7]
##  [1,] 0.1684203 0.7560663 0.1428054 0.21712503 0.4021982 0.9350485 0.7769184
##  [2,] 0.9102281 0.7298959 0.4758198 0.75077768 0.4260216 0.5895071 0.5781783
##  [3,] 0.2840570 0.8766687 0.2514208 0.84038227 0.3202349 0.4662432 0.3906964
##  [4,] 0.4371068 0.0777983 0.7917321 0.60220024 0.8334189 0.1330925 0.6348968
##  [5,] 0.6208476 0.3116342 0.8754070 0.23475172 0.5818232 0.6989925 0.8556058
##  [6,] 0.8975291 0.4376260 0.8037730 0.34314805 0.8095990 0.5067020 0.7351627
##  [7,] 0.6302503 0.3195848 0.9265502 0.11683922 0.7899073 0.8455880 0.9284658
##  [8,] 0.8267084 0.6553531 0.9125573 0.01083941 0.1901118 0.4004821 0.7740680
##  [9,] 0.2234593 0.5041576 0.2909868 0.72302158 0.4453708 0.5299002 0.9174636
## [10,] 0.1012189 0.9630807 0.5648862 0.18525646 0.3141297 0.5662604 0.7411664
##              [,8]       [,9]      [,10]
##  [1,] 0.3171213 0.44223778 0.98095037
##  [2,] 0.7171101 0.03980728 0.06987698
##  [3,] 0.6519952 0.17223613 0.55352070
##  [4,] 0.3053582 0.32631368 0.90106745
##  [5,] 0.8135643 0.02807054 0.53495305
##  [6,] 0.7820257 0.77271202 0.93244927
##  [7,] 0.6698715 0.95857661 0.25596077
##  [8,] 0.6938373 0.75480396 0.37906312
##  [9,] 0.4751127 0.67288952 0.12283416
## [10,] 0.9454266 0.27697711 0.27005991
```

```r
# Print the value of the element in the second row and first column of MT
value <- MT[2, 1]
print(value)
```

```
## [1] 0.9102281
```

```r
# Calculate the inner product between MT and M using a nested loop
N <- matrix(0, nrow = 10, ncol = 10)
for (i in 1:10) {
  for (j in 1:10) {
    N[i, j] <- sum(MT[i,] * M[,j])
  }
}
print(N)
```

```
##            [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]     [,8]
##  [1,] 3.565623 2.421434 2.623237 2.454212 2.863674 3.546303 3.222601 2.745393
##  [2,] 2.421434 3.535352 2.799265 2.379018 3.161892 3.501838 3.243759 2.991904
##  [3,] 2.623237 2.799265 2.852416 2.228553 2.544818 3.070893 2.543829 2.390135
##  [4,] 2.454212 2.379018 2.228553 3.313641 2.990848 3.809449 3.212620 2.644304
##  [5,] 2.863674 3.161892 2.544818 2.990848 3.812033 4.088760 3.883285 3.360187
##  [6,] 3.546303 3.501838 3.070893 3.809449 4.088760 5.291648 4.744129 4.171210
##  [7,] 3.222601 3.243759 2.543829 3.212620 3.883285 4.744129 5.005631 4.070125
##  [8,] 2.745393 2.991904 2.390135 2.644304 3.360187 4.171210 4.070125 3.936351
##  [9,] 2.573496 2.661320 2.428023 2.302221 2.605953 3.212779 3.302481 2.679694
```

```
## [10,] 2.784962 2.806990 2.638549 2.108118 3.035101 3.321238 3.299667 3.059951
##           [,9]     [,10]
##  [1,] 2.573496 2.784962
##  [2,] 2.661320 2.806990
##  [3,] 2.428023 2.638549
##  [4,] 2.302221 2.108118
##  [5,] 2.605953 3.035101
##  [6,] 3.212779 3.321238
##  [7,] 3.302481 3.299667
##  [8,] 2.679694 3.059951
##  [9,] 2.926032 2.595172
## [10,] 2.595172 3.303322
```

```r
# Calculate the same inner product using %*% operator
N_operator <- MT %*% M
print(N_operator)
```

```
##           [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]     [,8]
##  [1,] 3.565623 2.421434 2.623237 2.454212 2.863674 3.546303 3.222601 2.745393
##  [2,] 2.421434 3.535352 2.799265 2.379018 3.161892 3.501838 3.243759 2.991904
##  [3,] 2.623237 2.799265 2.852416 2.228553 2.544818 3.070893 2.543829 2.390135
##  [4,] 2.454212 2.379018 2.228553 3.313641 2.990848 3.809449 3.212620 2.644304
##  [5,] 2.863674 3.161892 2.544818 2.990848 3.812033 4.088760 3.883285 3.360187
##  [6,] 3.546303 3.501838 3.070893 3.809449 4.088760 5.291648 4.744129 4.171210
##  [7,] 3.222601 3.243759 2.543829 3.212620 3.883285 4.744129 5.005631 4.070125
##  [8,] 2.745393 2.991904 2.390135 2.644304 3.360187 4.171210 4.070125 3.936351
##  [9,] 2.573496 2.661320 2.428023 2.302221 2.605953 3.212779 3.302481 2.679694
## [10,] 2.784962 2.806990 2.638549 2.108118 3.035101 3.321238 3.299667 3.059951
##           [,9]     [,10]
##  [1,] 2.573496 2.784962
##  [2,] 2.661320 2.806990
##  [3,] 2.428023 2.638549
##  [4,] 2.302221 2.108118
##  [5,] 2.605953 3.035101
##  [6,] 3.212779 3.321238
##  [7,] 3.302481 3.299667
##  [8,] 2.679694 3.059951
##  [9,] 2.926032 2.595172
## [10,] 2.595172 3.303322
```

#Part 1.3

```r
# Get the current working directory
current_directory <- getwd()

# Print the current directory
print(current_directory)
```

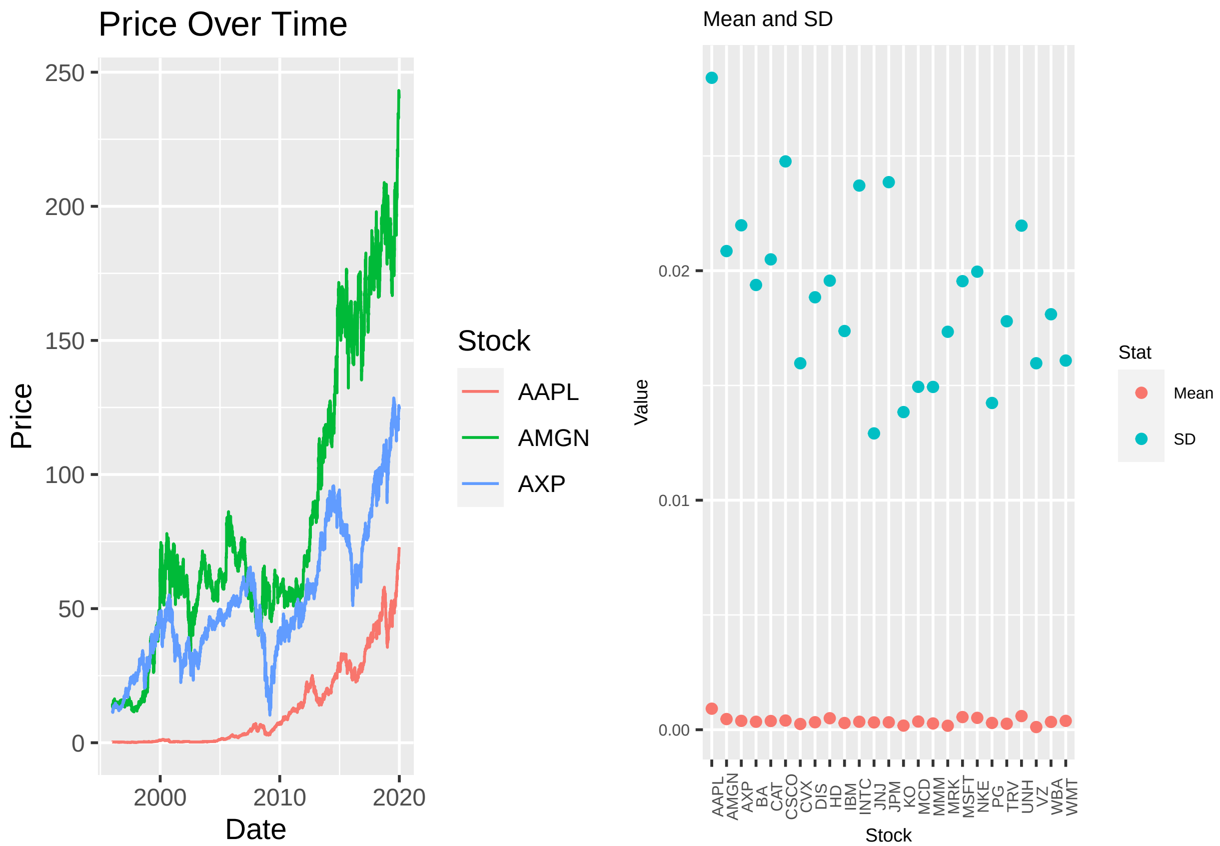
```
## [1] "G:/My Drive/1. DS/SEM 3/FE513"
```

```r
analyze_stock_data <- function(file_path) {
  # Load the CSV file
```

```r
  data <- read.csv(file_path, header = TRUE)
  data$X <- as.Date(data$X)
  names(data)[names(data) == "X"] <- 'Date'

  # Remove columns with NA values
  data_clean <- data[, colSums(is.na(data)) == 0]

  # Calculate daily log returns for each stock
  log_stocks <- lapply(data_clean[2:26], function(x) diff(log(x)))

  # Calculate mean returns for each stock
  mean_returns <- as.data.frame(sapply(X=log_stocks, FUN = mean))
  mean_returns <- cbind(newColName = rownames(mean_returns), mean_returns)
  colnames(mean_returns) <- c("Stock","Mean")

  # Calculate standard deviation of returns for each stock
  sd_returns <- as.data.frame(sapply(X=log_stocks, FUN = sd))
  sd_returns <- cbind(newColName = rownames(sd_returns), sd_returns)
  colnames(sd_returns) <- c("Stock","SD")

  # Merge mean and standard deviation data frames
  mean_sd_df <- merge(mean_returns, sd_returns, by = "Stock")

  # Load necessary libraries
  library(reshape2)
  library(ggpubr)

  # Reshape the data for plotting
  first_three <- data[, 1:4]
  df <- melt(first_three, id.vars = 'Date', variable.name = 'Stock', value.name = "Price")

  df2 <- melt(mean_sd_df, id.vars = 'Stock', variable.name = 'Stat', value.name = "Value")

  # Create line plot for stock prices over time
  line_g <- ggplot(data = df, aes(x = Date, y = Price, colour = Stock)) +
    geom_line() +
    ggtitle("Price Over Time")

  # Create scatter plot for mean and standard deviation
  point_g <- ggplot(data = df2, aes(x = Stock, y = Value, colour = Stat)) +
    geom_point() +
    ggtitle("Mean and SD") +
    theme(axis.text.x = element_text(angle = 90), text = element_text(size = 7))

  # Arrange and display both plots
  ggarrange(line_g, point_g)
}

# Call the function with the file path
result_plots <- analyze_stock_data("stock_data-1.csv")
```

```
## Loading required package: ggplot2
```

```r
# Display the resulting plots
print(result_plots)
```



#Part 1.3 BONUS

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

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

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

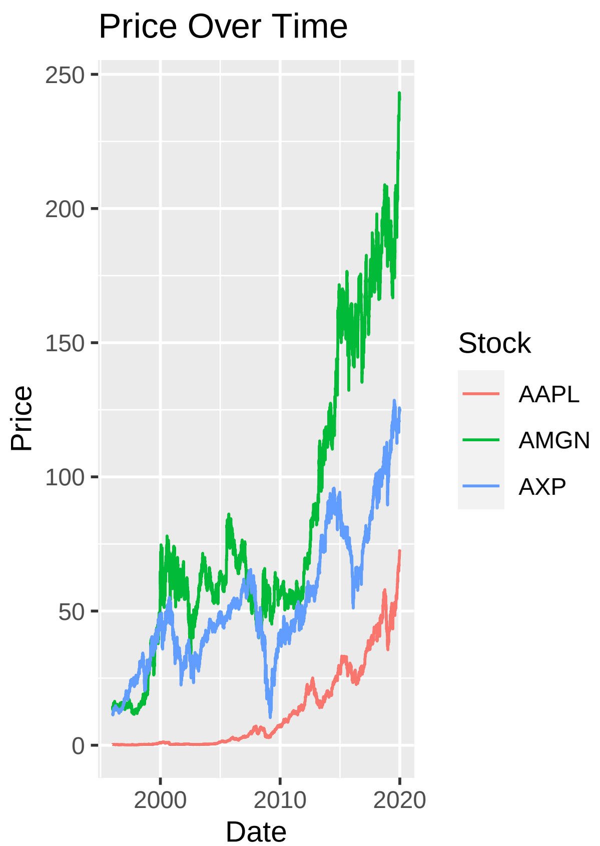
```r
library(tidyr)
```

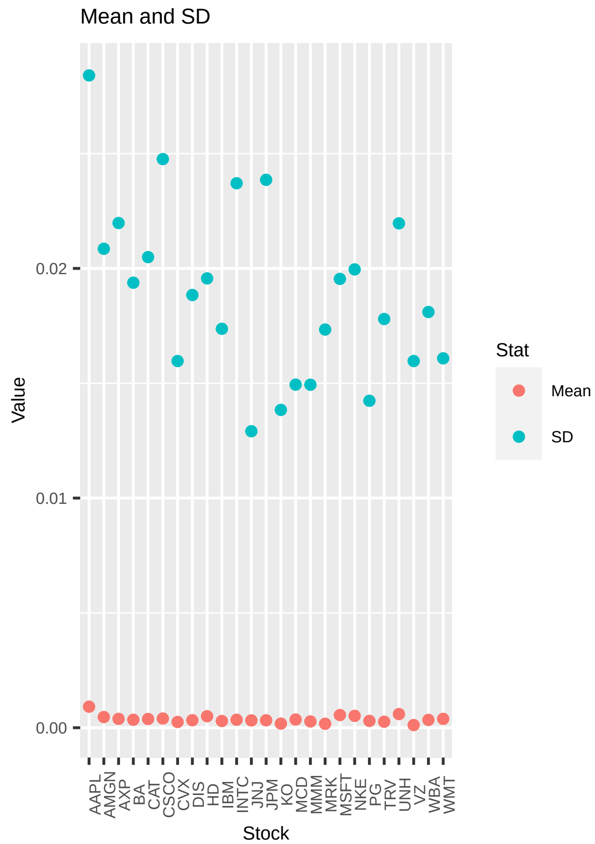
```
##
## Attaching package: 'tidyr'
```

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

```r
library(ggplot2)
library(reshape2)
library(ggpubr)

analyze_stock_data <- function(file_path) {
  # Load the CSV file and clean data
  data <- read.csv(file_path, header = TRUE) %>%
    mutate(Date = as.Date(X), .keep = "unused") %>%
    select(-contains("NA")) %>%
    select(Date, everything())

  # Calculate daily log returns for each stock
  log_stocks <- data %>%
    select(-Date) %>%
    lapply(function(x) diff(log(x)))

  # Calculate mean returns for each stock
  mean_returns <- log_stocks %>%
    sapply(mean) %>%
    data.frame(Stock = names(.), Mean = .)

  # Calculate standard deviation of returns for each stock
  sd_returns <- log_stocks %>%
    sapply(sd) %>%
    data.frame(Stock = names(.), SD = .)

  # Merge mean and standard deviation data frames
  mean_sd_df <- merge(mean_returns, sd_returns, by = "Stock")

  # Reshape the data for plotting
  first_three <- data[, 1:4] %>%
    melt(id.vars = "Date", variable.name = "Stock", value.name = "Price")

  df2 <- melt(mean_sd_df, id.vars = "Stock", variable.name = "Stat", value.name = "Value")
  df2 <- na.omit(df2)  # Remove rows with missing values

  # Create line plot for stock prices over time
  line_g <- ggplot(data = first_three, aes(x = Date, y = Price, colour = Stock)) +
    geom_line() +
    ggtitle("Price Over Time")

  # Create scatter plot for mean and standard deviation
  point_g <- ggplot(data = df2, aes(x = Stock, y = Value, colour = Stat)) +
    geom_point() +
    ggtitle("Mean and SD") +
    theme(axis.text.x = element_text(angle = 90), text = element_text(size = 7))
  point_g <- na.omit(point_g)

  # Arrange and display both plots
  ggarrange(line_g, point_g)
```

```
}

# Call the function with the file path
result_plots <- analyze_stock_data("stock_data-1.csv")

# Display the resulting plots
print(result_plots)
```





#Part 2

```
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

##
```

```
## ######################### 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: 'xts'
```

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

```
## Loading required package: TTR
```

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

```
# Part 2.1: Download AMZN stock data
getSymbols(c("AMZN"), from = '2021-01-01',
           to = "2021-09-01", warnings = FALSE,
           auto.assign = TRUE)
```

```
## [1] "AMZN"
```

```
# Convert AMZN data to a data frame
AMZN <- data.frame(AMZN)

# Save the data to a CSV file
write.csv(AMZN, file = "AMZN_stock_data_2021.csv")
head(AMZN)
```

```
##            AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
## 2021-01-04  163.5000  163.6000  157.201   159.3315    88228000      159.3315
## 2021-01-05  158.3005  161.1690  158.253   160.9255    53110000      160.9255
## 2021-01-06  157.3240  159.8755  156.558   156.9190    87896000      156.9190
## 2021-01-07  157.8500  160.4270  157.750   158.1080    70290000      158.1080
## 2021-01-08  159.0000  159.5320  157.110   159.1350    70754000      159.1350
## 2021-01-11  157.4005  157.8190  155.500   155.7105    73668000      155.7105
```

```r
# Part 2.2: Calculate log returns
  AMZN$log_returns <- c(0, diff(log(AMZN$AMZN.Close), lag = 1))

  # Part 2.3: Calculate mean, median, and standard deviation
  mean_return <- mean(AMZN$log_returns)
  mean_return
```

```
## [1] 0.0005114871
```

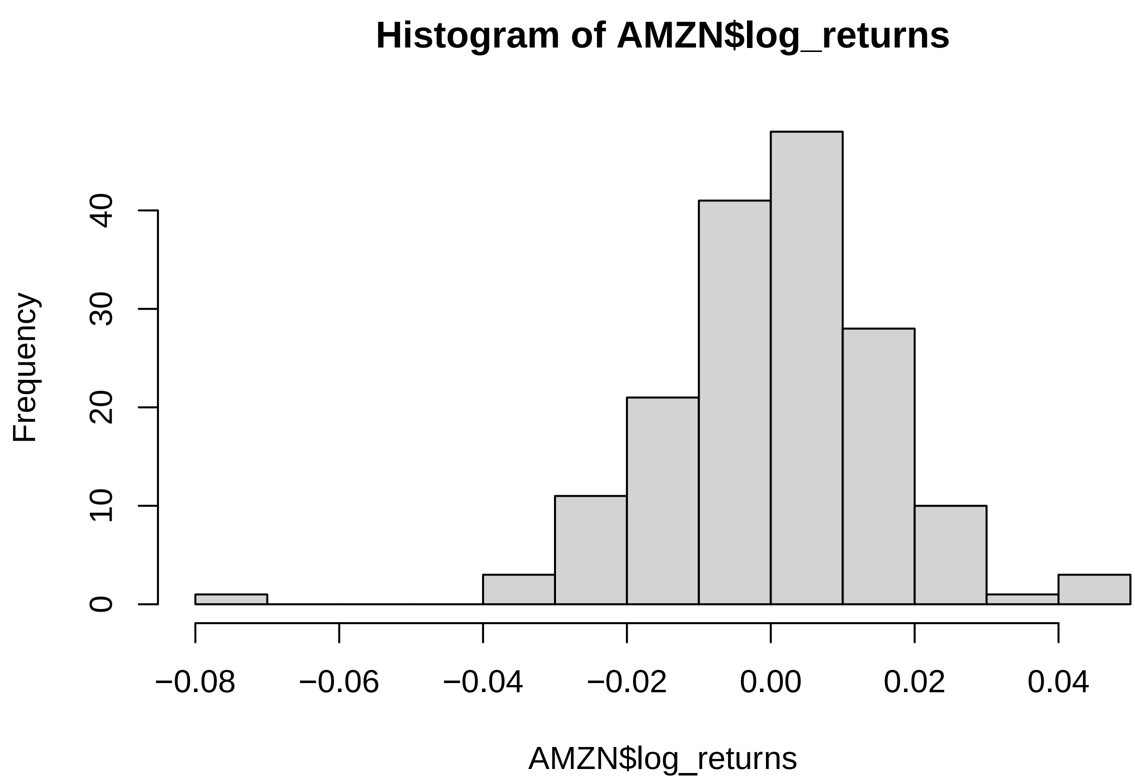
```r
median_return <- median(AMZN$log_returns)
  median_return
```

```
## [1] 0.001557905
```

```r
sd_return <- sd(AMZN$log_returns)
  sd_return
```

```
## [1] 0.01539873
```

```r
# Part 2.4: Create a histogram
  hist(AMZN$log_returns)
```



```r
# Part 2.5: Calculate the range of log returns and create Log.Range column
  amzn_range <- range(AMZN$log_returns)
  range_diff <- amzn_range[2] - amzn_range[1]
  range_diff
```

```
## [1] 0.1245226
```

```r
AMZN$Log.Range <- cut(AMZN$log_returns, breaks = seq(from = -0.07, to = 0.05, by = 0.005))
  log_range_count <- length(AMZN$Log.Range[AMZN$Log.Range == "(0.01,0.015]"])

log_range_count
```

```
## [1] 24
```