Final Exam FE513

Shubham Narkhede

2023-12-12

```
if(!require("quantmod")){ # check for package existence
  install.packages("quantmod")
}
```

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

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

```
library("quantmod")
```

```
if(!require("zoo")){ # check for package existence
  install.packages("zoo")
}
library("zoo")
```

```
if(!require("ggplot2")){ # check for package existence
  install.packages("ggplot2")
}
```

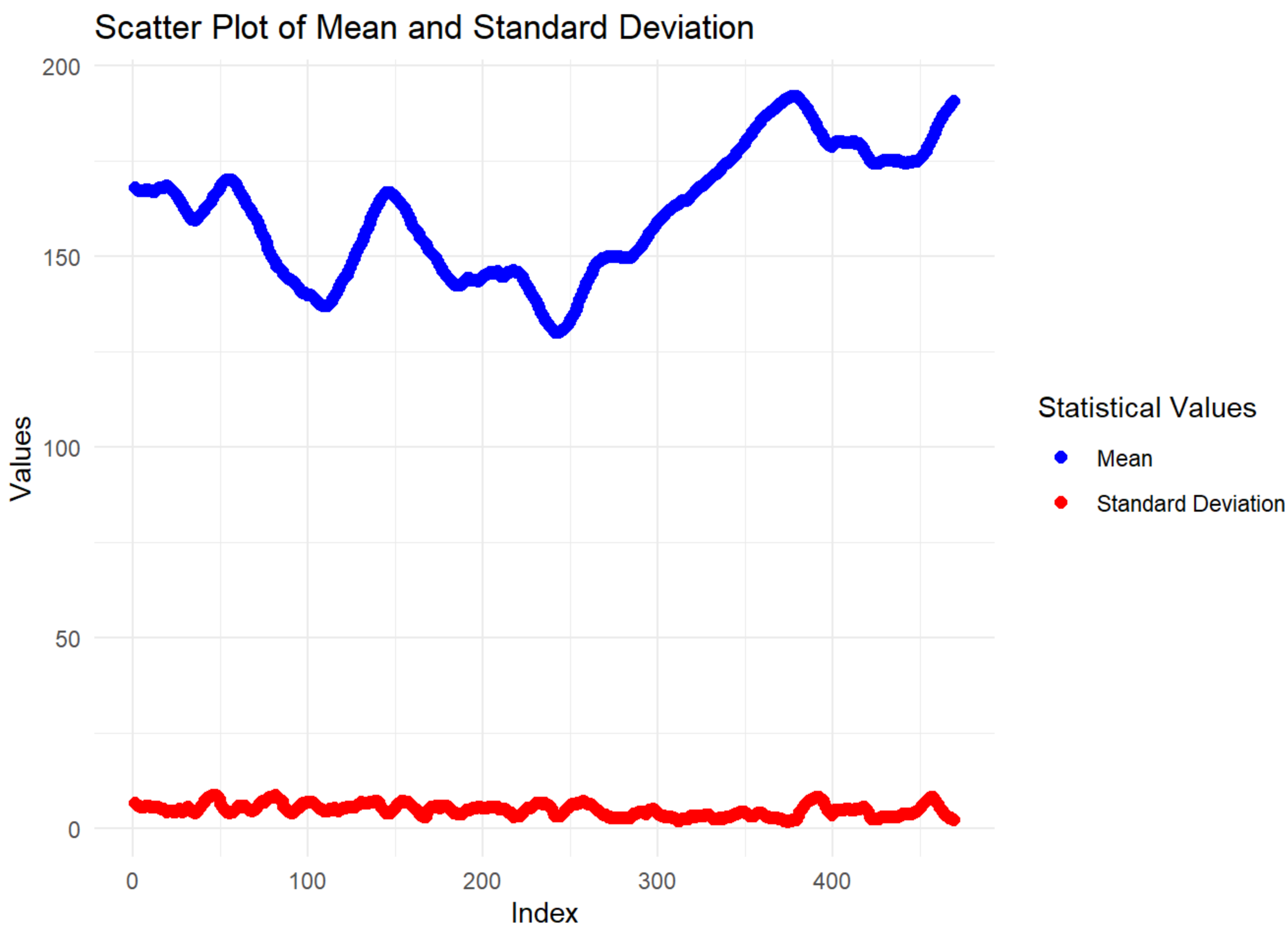
```
## Loading required package: ggplot2
```

```
library(ggplot2)
```

Question 2

```
get_stock_statistics <- function(ticker, start_date, end_date, rolling_window) {
  # Step 1: Download daily stock data
  start_date = as.Date(start_date)
  end_date = as.Date(end_date)
  getSymbols(ticker, src = "yahoo", from = start_date, to = end_date)
  stocks = get(ticker)

  # Step 2: Get the adjusted close price
  stocks = stocks[, 6]

  # Step 3: Perform rolling window estimation
  count = length(stocks)
  avg = vector()
  standard_dev = vector()

  for (i in 1:(count - rolling_window + 1)) {
    arr = stocks[i:(i + rolling_window - 1)]
    avg = c(avg, mean(arr))
    standard_dev = c(standard_dev, sd(arr))
  }

  # Step 4: Store the statistical result into a dataframe
  df = data.frame(Index = seq_along(avg), Mean = avg, Standard_Deviation = standard_dev)

  # Step 5: Plot the statistical dataframe using a scatter plot
  gg <- ggplot(df, aes(x = Index)) +
    geom_point(aes(y = Mean, color = "Mean"), size = 2) +
    geom_point(aes(y = Standard_Deviation, color = "Standard Deviation"), size = 2) +
    labs(title = "Scatter Plot of Mean and Standard Deviation",
         x = "Index",
         y = "Values") +
    scale_color_manual(name = "Statistical Values",
                       values = c("Mean" = "blue", "Standard Deviation" = "red")) +
    theme_minimal()

  # Explicitly print the plot
  print(gg)

  # Step 6: Return the statistical dataframe
  return(df)
}
```

```
# Step 7: Test the function with suitable parameters
start_date <- "2022-01-01"
end_date <- Sys.Date()
stock_ticker <- "AAPL"
window_size <- 20

result_df <- get_stock_statistics(stock_ticker, start_date, end_date, window_size)
```




```
head(result_df,10)
```

```
##    Index     Mean Standard_Deviation
## 1      1 167.9426           6.684206
## 2      2 167.5768           6.172897
## 3      3 167.3860           5.897305
## 4      4 167.2861           5.814565
## 5      5 167.3163           5.831221
## 6      6 167.3019           5.824011
## 7      7 167.4434           5.932663
## 8      8 167.5138           6.011213
## 9      9 167.3561           5.884624
## 10    10 167.1912           5.845491
```

Question 3

```
if(!require("RPostgreSQL")){ # check for package existence
  install.packages("RPostgreSQL")
}
```

```
## Loading required package: RPostgreSQL
```

```
## Loading required package: DBI
```

```
library("RPostgreSQL")
```

```
library(RPostgreSQL)
library(DBI)

# 1. Make a connection to your local PostgreSQL database.
con <- dbConnect(RPostgres::Postgres(),
                 dbname = "FinalExam",
                 host = "localhost",
                 port = 5432,
                 user = "postgres",
                 password = '123')

# 2. Query the PostgreSQL database via API to get the original bank data.
query <- "SELECT * FROM bank_data"
bank_data_df <- dbGetQuery(con, query)
head(bank_data_df)
```

```
##      id       date asset liability idx
## 1 23373 2002-09-30 95914     87304   1
## 2 23376 2002-12-31 95937     87453   2
## 3 23376 2002-03-31 83335     75939   3
## 4 23376 2002-06-30 84988     77125   4
## 5 23376 2002-09-30 90501     82248   5
## 6   234 2002-12-31 56866     49406   6
```

```
if(!require("dplyr")){ # check for package existence
  install.packages("dplyr")
}
```

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

```
library("dplyr")
```

```
# 3. Calculate asset growth rate for each quarter and each bank.
bank_data_df$date <- as.Date(bank_data_df$date)

bank_data_df <- bank_data_df[order(bank_data_df$id, bank_data_df$date), ]  # Ensure data is sort
ed by bank id and date


# Calculate asset growth rate excluding the first quarter
bank_data_df_growth_rate <- bank_data_df %>%
  arrange(id, date) %>%
  group_by(id) %>%
  mutate(
    previous_asset = lag(asset),
    asset_growth_rate = (asset - previous_asset) / previous_asset
  ) %>%
  filter(!is.na(asset_growth_rate))

# Display the first 10 rows of the resulting data frame
head(bank_data_df_growth_rate, 10)
```

```
## # A tibble: 10 × 7
## # Groups:   id [4]
##       id date          asset liability   idx previous_asset asset_growth_rate
##    <int> <date>        <int>     <int> <int>          <int>             <dbl>
##  1     9 2002-06-30   361953    332900 20913         348727            0.0379
##  2     9 2002-09-30   383246    352456 20914         361953            0.0588
##  3     9 2002-12-31   371812    340365 20911         383246           -0.0298
##  4    14 2002-06-30 73600000  69200000 27335       68600000            0.0729
##  5    14 2002-09-30 72800000  68200000 27336       73600000           -0.0109
##  6    14 2002-12-31 79600000  74500000 27333       72800000            0.0934
##  7    28 2002-06-30    12049      5354  3938          14340           -0.160
##  8    28 2002-09-30    12474      5543  3939          12049            0.0353
##  9    35 2002-06-30   492046    457116 12624         471056            0.0446
## 10    35 2002-09-30   503401    467080 12625         492046            0.0231
```

```
# 4. Export the dataframe of Q 3.3 to the PostgreSQL database via API
# Delete the existing table if it exists
dbExecute(con, "DROP TABLE IF EXISTS asset_growth_rate_table")
```

```
## [1] 0
```

```
dbWriteTable(con, name = "asset_growth_rate_table", value = bank_data_df_growth_rate, row.names
= FALSE, overwrite = TRUE)

query <- "SELECT * FROM asset_growth_rate_table"
asset_growth_rate_df <- dbGetQuery(con, query)
head(asset_growth_rate_df)
```

```
##   id       date    asset liability   idx previous_asset asset_growth_rate
## 1  9 2002-06-30   361953    332900 20913         348727        0.03792652
## 2  9 2002-09-30   383246    352456 20914         361953        0.05882808
## 3  9 2002-12-31   371812    340365 20911         383246       -0.02983462
## 4 14 2002-06-30 73600000  69200000 27335       68600000        0.07288630
## 5 14 2002-09-30 72800000  68200000 27336       73600000       -0.01086957
## 6 14 2002-12-31 79600000  74500000 27333       72800000        0.09340659
```

```
# Close the database connection
dbDisconnect(con)
```