Stat 432 Homework 1

Assigned: Aug 26, 2024; Due: 11:59 PM CT, Sep 5, 2024

Instruction

Please remove this section when submitting your homework.

Students are encouraged to work together on homework and/or utilize advanced AI tools. However, **sharing, copying, or providing any part of a homework solution or code to others** is an infraction of the University's rules on Academic Integrity (https://studentcode.illinois.edu/article1/part4/1-401/). Any violation will be punished as severely as possible. Final submissions must be uploaded to Gradescope (https://www.gradescope.com/courses/570816). No email or hard copy will be accepted. For **late submission policy and grading rubrics** (https://teazrq.github.io/stat432/syllabus.html), please refer to the course website.

- You are required to submit the rendered file `HWx_yourNetID.pdf`. For example, `HW01_rqzhu.pdf`. Please note that this must be a `.pdf` file. `.html` format **cannot** be accepted. Make all of your `R` code chunks visible for grading.
- Include your Name and NetID in the report.
- If you use this file or the example homework `.Rmd` file as a template, be sure to **remove this instruction** section.
- Make sure that you **set seed** properly so that the results can be replicated if needed.
- For some questions, there will be restrictions on what packages/functions you can use. Please read the requirements carefully. As long as the question does not specify such restrictions, you can use anything.
- **When using AI tools**, try to document your prompt and any follow-up prompts that further modify or correct the answer. You are also required to briefly comment on your experience with it, especially when it's difficult for them to grasp the idea of the question.
- **On random seed and reproducibility**: Make sure the version of your `R` is $\geq 4.0.0$. This will ensure your random seed generation is the same as everyone else. Please note that updating the `R` version may require you to reinstall all of your packages.

Question 1 (Multivariate Normal Distribution)

This question is about playing with AI tools for generating multivariate normal random variables. Let X_i, $i = 1, \ldots, n$ be i.i.d. multivariate normal random variables with mean μ and covariance matrix Σ, where

$$\mu = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \text{and} \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}.$$

Write `R` code to perform the following tasks. Please try to use AI tools as much as possible in this question.

a. [10 points] Generate a set of $n = 2000$ observations from this distribution. Only display the first 5 observations in your `R` output. Make sure set random seed $= 1$ in order to replicate the result. Calculate the sample covariance matrix of the generated data and compare it with the true covariance matrix Σ.

```
set.seed(1)
n <- 2000
mu <- c(1, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), nrow = 2, ncol = 2)
X <- MASS::mvrnorm(n, mu, sigma)
head(X, 5)
```

```
##            [,1]     [,2]
## [1,]  0.9005499 1.014400
## [2,]  2.1201672 1.197912
## [3,] -0.5335260 2.086175
## [4,]  2.1219187 3.641189
## [5,]  1.3132871 2.257437
```

```
cov(X)
```

```
##           [,1]      [,2]
## [1,] 1.0443799 0.5392157
## [2,] 0.5392157 1.1045078
```

b. [10 points] If you used VS Code and AI tools to perform the previous question, then they will most likely suggest using the `mvrnorm` function from the `MASS` package. However, there are alternative ways to complete this question. For example, you could first generate n standard normal random variables, and then transform them to the desired distribution. Write down the mathematical formula of this approach in Latex, and then write `R` code to implement this approach. Only display the first 5 observations in your `R` output. Validate your approach by computing the sample covariance matrix of the generated data and compare it with the true covariance matrix Σ. Please note that you **should not use** the `mvrnorm` function anymore in this question.

```
set.seed(1)
Z <- matrix(rnorm(n*2), n, 2)
X <- sweep(Z %*% chol(sigma), 2, mu, "+")
head(X, 5)
```

```
##           [,1]      [,2]
## [1,] 0.3735462 0.9193450
## [2,] 1.1836433 0.4271001
## [3,] 0.1643714 2.9848877
## [4,] 2.5952808 3.2473413
## [5,] 1.3295078 2.1163864
```

```
cov(X)
```

```
##           [,1]      [,2]
## [1,] 1.0757730 0.5679505
## [2,] 0.5679505 1.1018494
```

c. [10 points] Write an `R` function called `mymvnorm` that takes the following arguments: `n`, `mu`, `sigma`. The function should return a matrix of dimension $n \times p$, where p is the length of `mu`. The function should generate n observations from a multivariate normal distribution with mean `mu` and covariance matrix `sigma`. You should not use the `mvrnorm` function in your code. Instead, use the logic you wrote in part b) to generate the data. Again, validate your result by calculating the sample covariance matrix of the generated data and compare to Σ. Also, when setting seed correctly, your answer in this question should be identical to the one in part b).

```r
mymvnorm <- function(n, m, S) {
  p <- length(m)
  Z <- matrix(rnorm(n*p), n, p)
  X <- sweep(Z %*% chol(S), 2, m, "+")
  return(X)
}

set.seed(1)
X <- mymvnorm(n, m = mu, S = sigma)
head(X, 5)
```

```
##           [,1]      [,2]
## [1,] 0.3735462 0.9193450
## [2,] 1.1836433 0.4271001
## [3,] 0.1643714 2.9848877
## [4,] 2.5952808 3.2473413
## [5,] 1.3295078 2.1163864
```

```r
cov(X)
```

```
##           [,1]      [,2]
## [1,] 1.0757730 0.5679505
## [2,] 0.5679505 1.1018494
```

The logic of this approach can be explained by the following:

$$\begin{aligned} \textbf{Given}: &\quad \mathbf{Z} \sim \square(\mathbf{0}, \mathbf{I}) \\ \textbf{Cholesky Decomposition}: &\quad \mathbf{L} = \text{chol}(\mathbf{\Sigma}) \\ \textbf{Transformation}: &\quad \mathbf{X} = \mathbf{Z}\mathbf{L} + \mu \\ \textbf{Mean of X}: &\quad \mathbb{E}[\mathbf{X}] = \mathbb{E}[\mathbf{ZL}] + \mu = \mathbf{0L} + \mu = \mu \\ \textbf{Covariance of X}: &\quad \text{Cov}(\mathbf{X}) = \mathbf{L}\text{Cov}(\mathbf{Z})\mathbf{L}^T = \mathbf{LIL}^T = \mathbf{\Sigma} \end{aligned}$$

Since linear combinations of normal random variables are still normal, then $\mathbf{X} \sim \square(\mu, \mathbf{\Sigma})$.

d. [10 points] If you used any AI tools during the first three questions, write your experience here. Specifically, what tool(s) did you use? **What prompt was used**? Did the tool suggested a corrected answer to your question? If not, which part was wrong? How did you corrected their mistakes (e.g modifying your prompt)?

I tried to complete these questions in two platforms. In VS Code using Copilot, the program did not suggest the correct code. For example, in Question b), it still suggested using the `mvrnorm` function to generate independent normal random variables, which can be done using the `rnorm` function instead. Furthermore, the orientation of the data matrix was wrong.

In GPT-4, I used the prompt

> **Can you write me an R code that generates 2-dimensional normal distribution with mean 0 and covariance matrix with diagonal elements 1 and off diagonal elements 0.5.**

and follow-up prompts to get the latex code

Can you write this formula in latex form?

Can you instead just give me the latex code so that I can copy them into my own file?

gave the correct `R` code and latex formula to complete the question. But I did not use their code. I wrote my own code with the `sweep` function, which I am more familiar with.

Question 2 (Data Manipulation and Plots)

The following question practices data manipulation and summary statistics. Our goal is to write a function that calculates the price gap between any two given dates. Load the `quantmod` package and obtain the `AAPL` data (apple stock price).

```
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
```

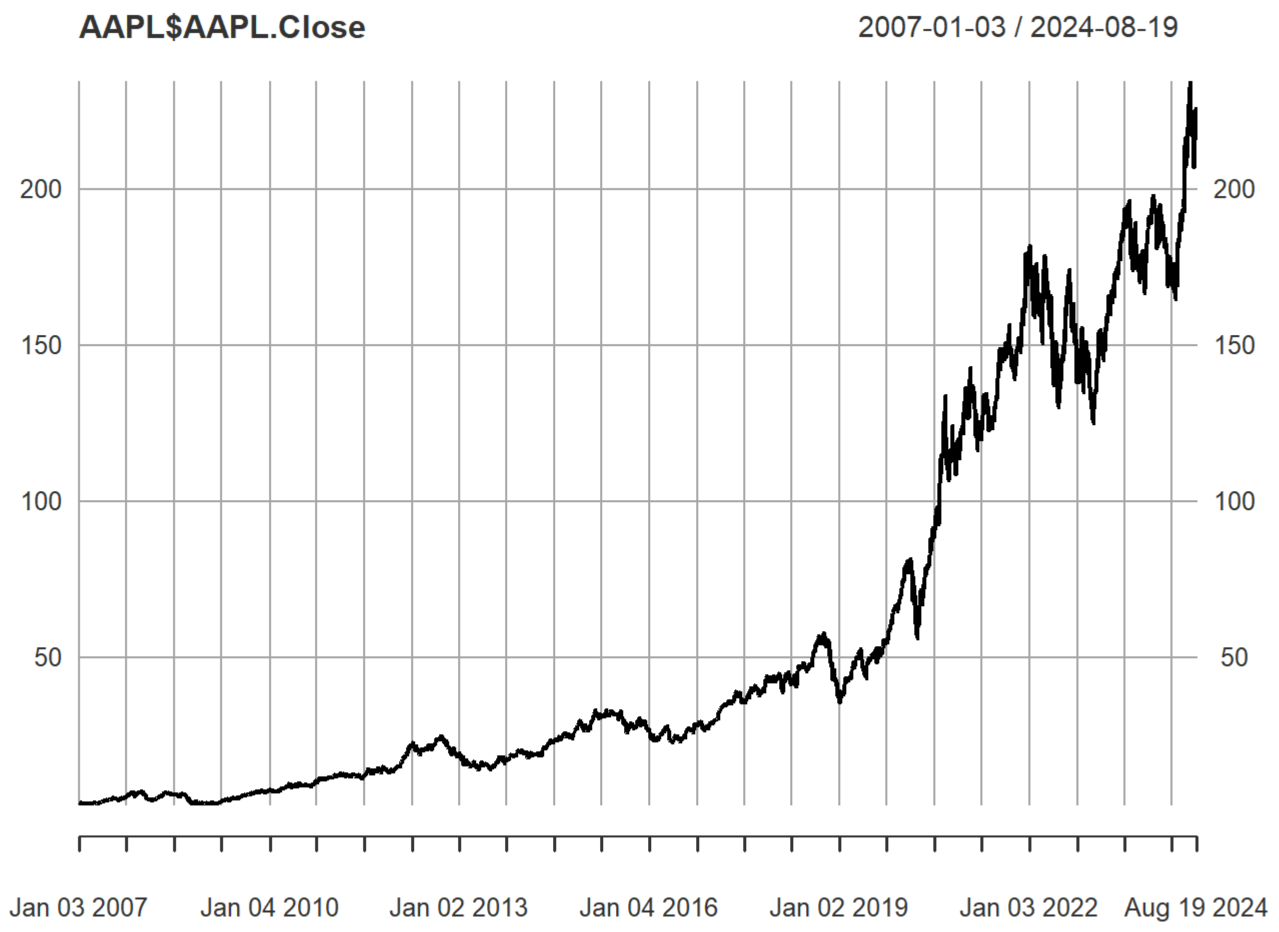
```
## Loading required package: TTR
```

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

```
getSymbols("AAPL")
```

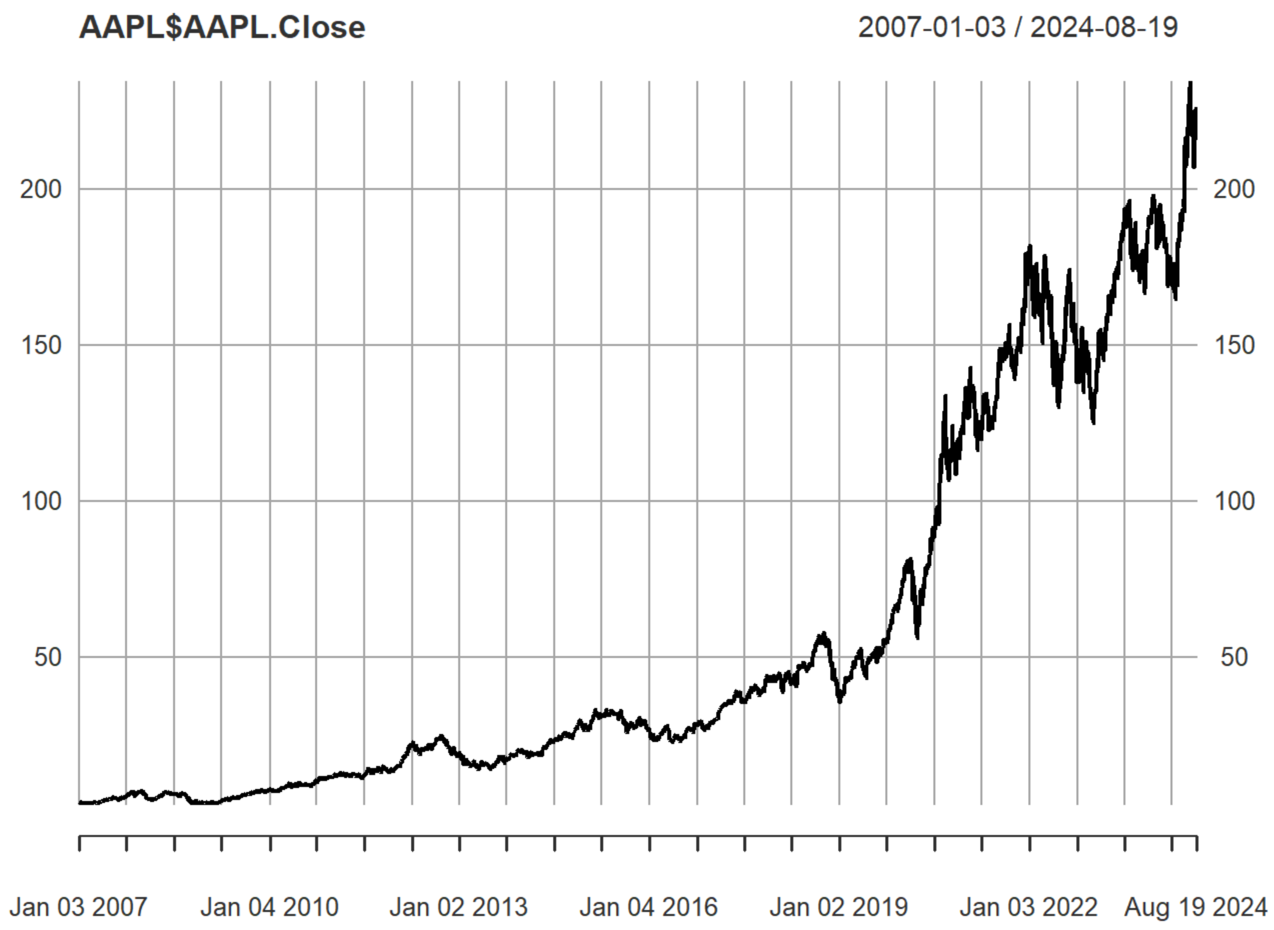
```
## [1] "AAPL"
```

```
plot(AAPL$AAPL.Close, pch = 19)
```

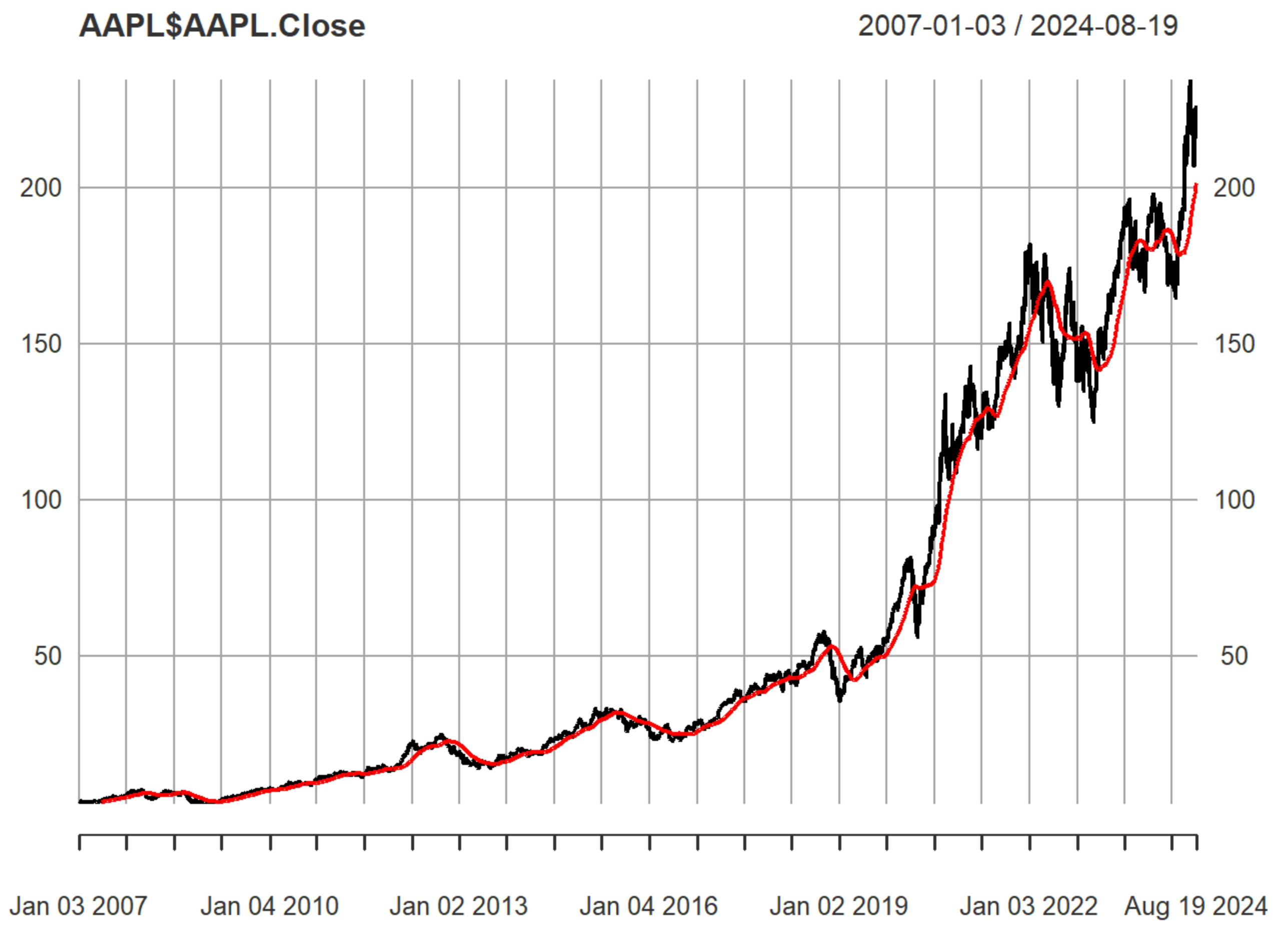


a. [20 points] Calculate a 90-day moving average of the closing price of `AAPL` and plot it on the same graph. Moving average means that for each day, you take the average of the previous 90 days. Please do this in two ways: 1) there is a built-in function called `SMA` in the `quantmod` package; 2) write your own function to calculate the moving average. For both questions, you can utilize AI tools to help you write the code.

```
AAPL$MA90 <- SMA(Cl(AAPL), 90)
plot(AAPL$AAPL.Close, pch = 19)
```

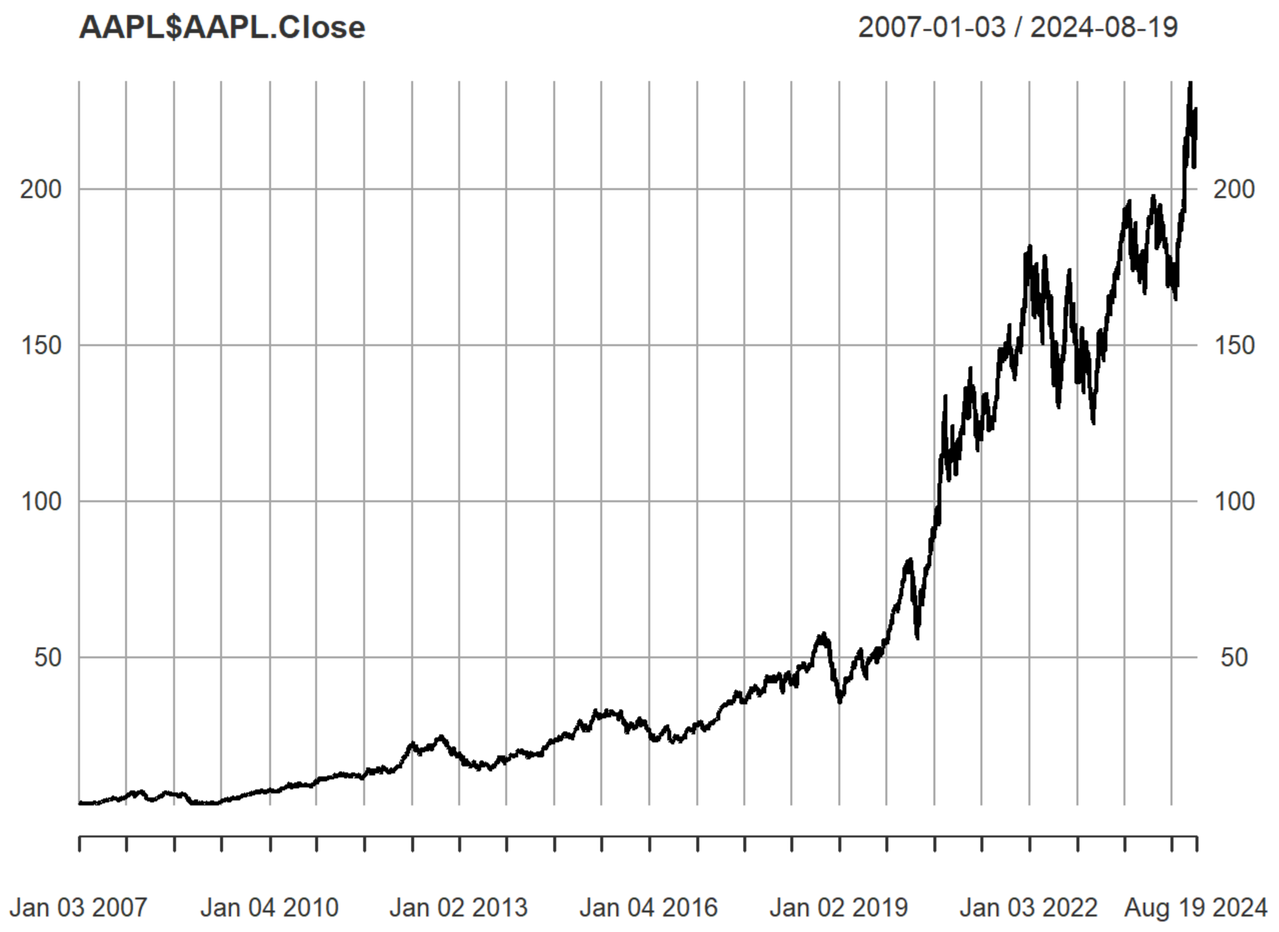


```
lines(AAPL$MA90, col = "red", lwd = 2)
```

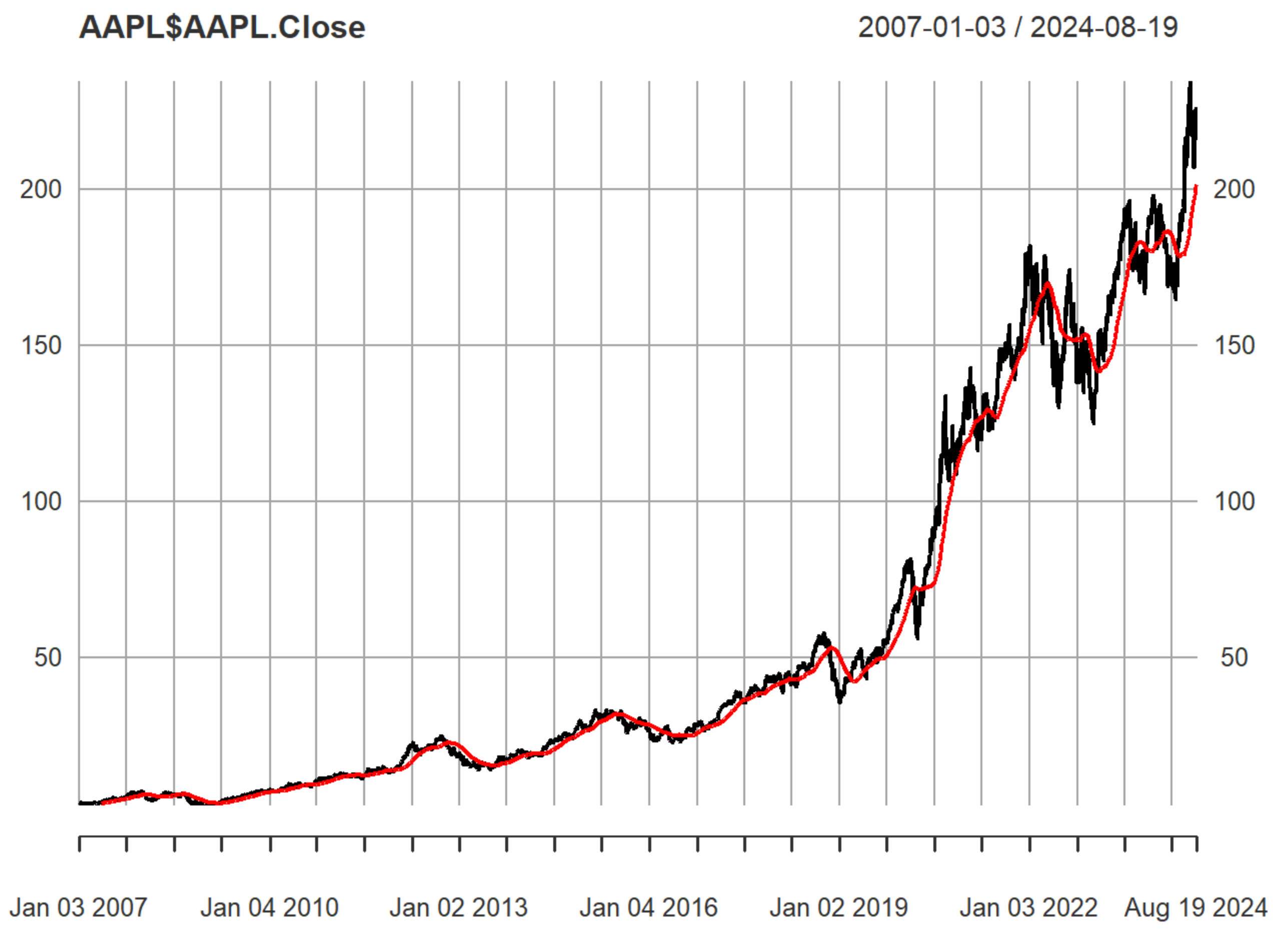


```r
moving_average <- function(x, window) {
  # Compute the moving average of x with window size = window
  n <- length(x)
  ma <- rep(NA, n)
  for (i in window:n) {
    ma[i] <- mean(x[(i-window+1):i])
  }
  return(ma)
}

AAPL$MA90 <- moving_average(Cl(AAPL), 90)
plot(AAPL$AAPL.Close, pch = 19)
```



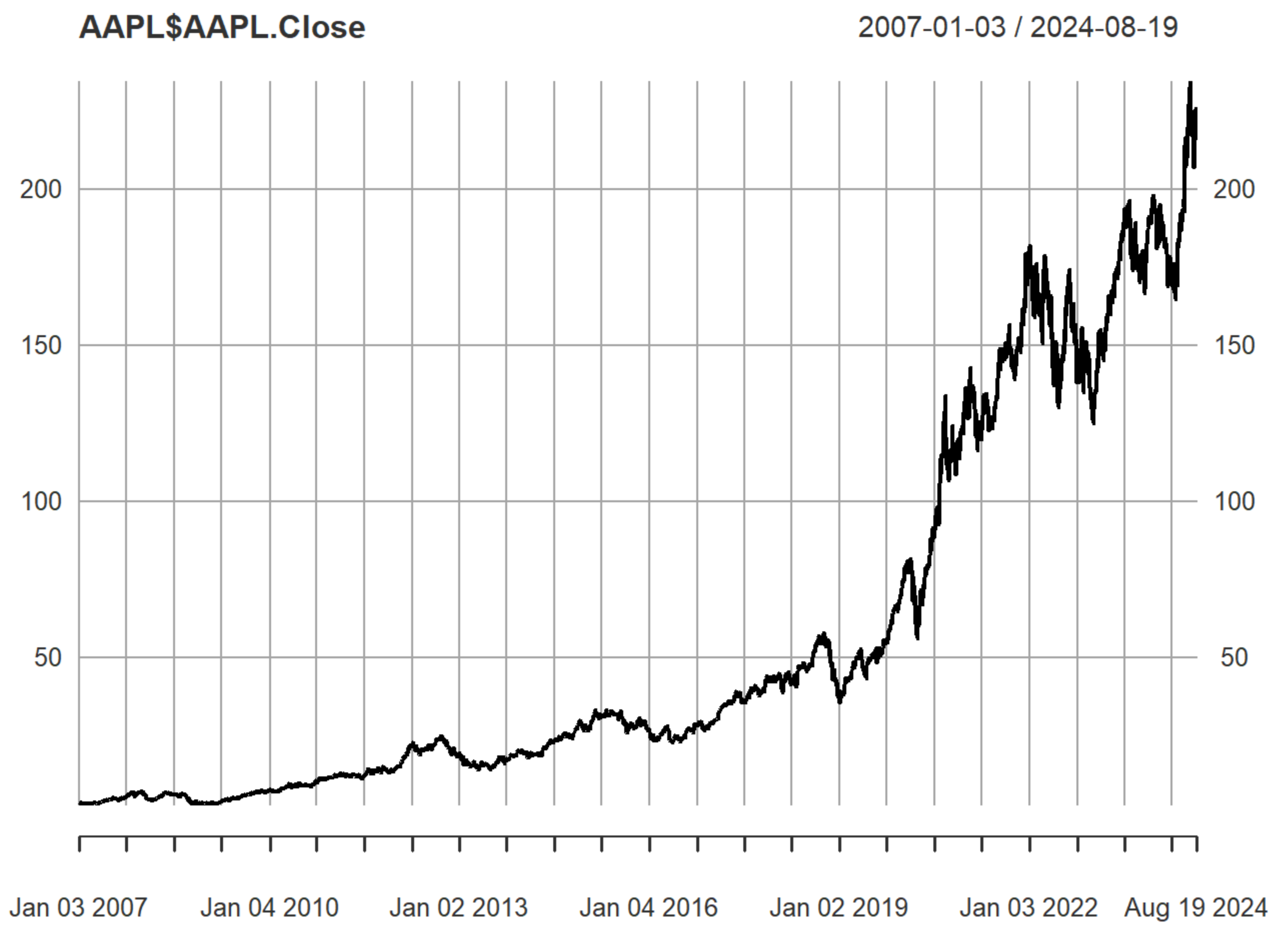
```r
lines(AAPL$MA90, col = "red", lwd = 2)
```



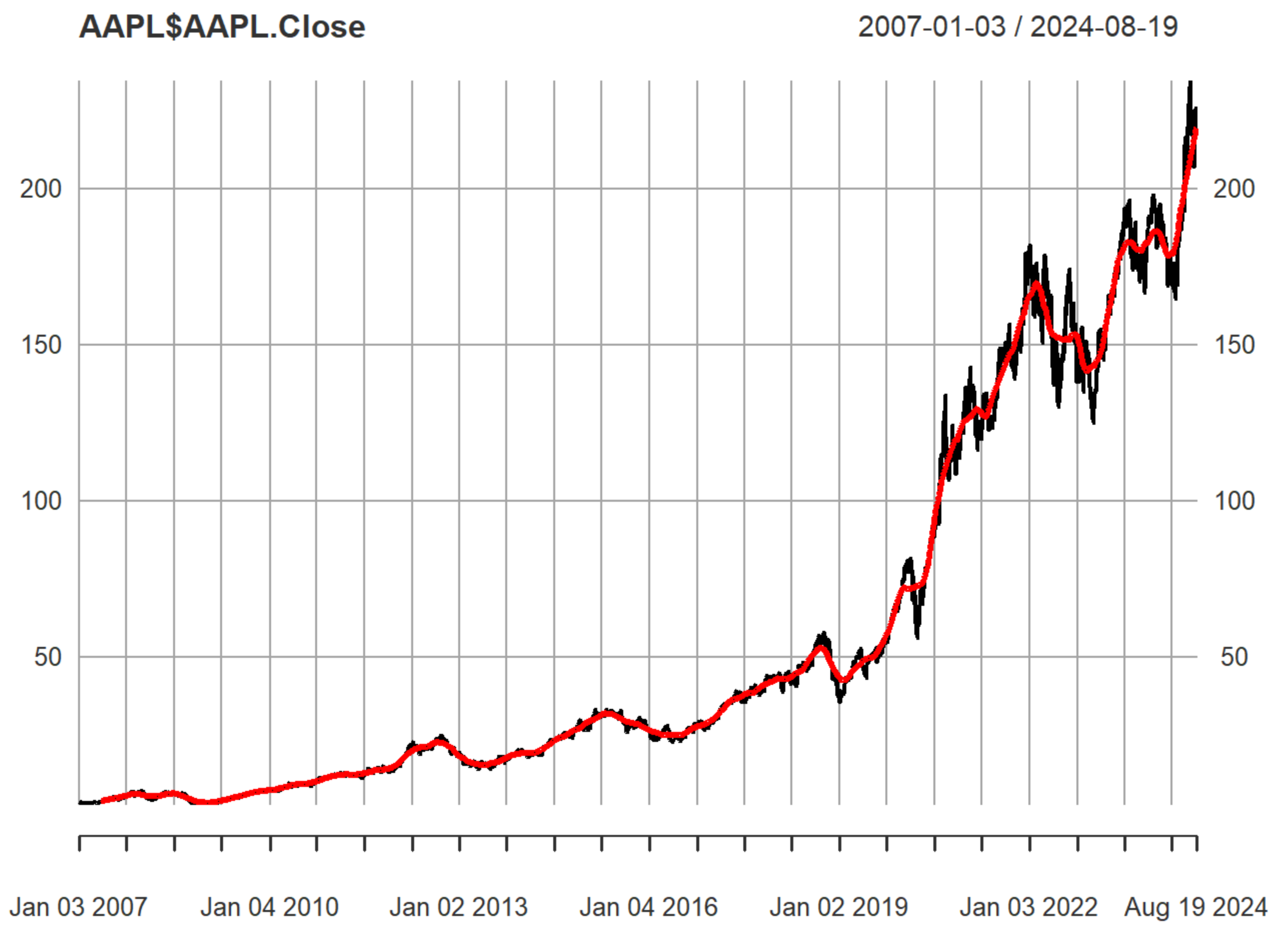
b. [15 points] I have an alternative way of writing this function.

```
my_average <- function(x, window) {
  # Compute the moving average of x with window size = window
  n <- length(x)
  ma <- rep(NA, n)
  for (i in window:n) {
    myinterval = (i-window/2):(i + window/2)
    myinterval = myinterval[myinterval > 0 & myinterval <= n]
    ma[i] <- mean( x[ myinterval ] )
  }
  return(ma)
}

AAPL$MA90 <- my_average(Cl(AAPL), 90)
plot(AAPL$AAPL.Close, pch = 19)
```



```
lines(AAPL$MA90, col = "red", lwd = 3)
```



Can you comment on the difference of these two functions? Do you think my line is a good choice when it is used for predicting future prices? Which one do you prefer and why.

Although my new function is approximating the observed stock price better, it is not a good idea to use it. The reason is that when we use the information for prediction, the moving average is supposed to be a **lagging indicator**. That is, it is supposed to be calculated using only the past information. However, my new function is using the future information to calculate the moving average. This is not a good idea because we do not know the future.

Question 3 (Read/write Data)

a. [10 points] The `ElemStatLearn` package [CRAN link (https://cran.r-project.org/web/packages/ElemStatLearn/index.html)] is an archived package. Hence, you cannot directly install it using the `install.packages()` function. Instead, you may install an older version of it by using the `install_github()` function from the `devtools` package. Install the `devtools` package and run the find the code to install the `ElemStatLearn` package.

```r
library(devtools)
devtools::install_github("cran/ElemStatLearn")

# alternatively, you can do
install.packages("ElemStatLearn", version = "2015.6.26.2",
                 repos = "http://cran.us.r-project.org")
```

b. [15 Points] Load the `ElemStatLearn` package and obtain the `ozone` data. Save this data into a `.csv` file, and then read the data back from that file into `R`. Print out the first 5 observations to make sure that the new data is the same as the original one.

We first get the `ozone` data and save it in a `.csv` file.

```r
library(ElemStatLearn)
data(ozone)
write.csv(ozone, file = "ozone.csv")
```

We then load the data back into `R`:

```r
ozone_new = read.csv("ozone.csv")
```

However, these two files are different because the `read.csv()` function will treat the first column (observation IDs) as a new variable.

```r
head(ozone_new)
```

	X	ozone	radiation	temperature	wind
	<int>	<dbl>	<int>	<int>	<dbl>
1	1	41	190	67	7.4
2	2	36	118	72	8.0
3	3	12	149	74	12.6
4	4	18	313	62	11.5
5	5	23	299	65	8.6
6	6	19	99	59	13.8

6 rows

To prevent this issue, we will specify the argument `row.names = 1`. This leads to the correct data.

```
ozone_new = read.csv("ozone.csv", row.names = 1)
head(ozone_new)
```

	ozone	radiation	temperature	wind
	<dbl>	<int>	<int>	<dbl>
1	41	190	67	7.4
2	36	118	72	8.0
3	12	149	74	12.6
4	18	313	62	11.5
5	23	299	65	8.6
6	19	99	59	13.8

6 rows

For this question, my AI tool (Copilot) did not suggest the correct code since it did not realize that R will also read the row names as a new variable. Hence, after doing some research, I found that adding `row.names = 1` solves the problem. This was the code originally suggested by Copilot.

```
library(ElemStatLearn)
data(ozone)
write.csv(ozone, "ozone.csv")
ozone2 <- read.csv("ozone.csv")
head(ozone2, 5)
```

	X	ozone	radiation	temperature	wind
	<int>	<dbl>	<int>	<int>	<dbl>
1	1	41	190	67	7.4
2	2	36	118	72	8.0
3	3	12	149	74	12.6
4	4	18	313	62	11.5
5	5	23	299	65	8.6

5 rows