STAT432_HW1

Qianhua Zhou(qz33)

2024-09-02

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.

```r
library(MASS)
set.seed(1)
n <- 2000
mu <- c(1, 2)
Sigma <- matrix(c(1, 0.5, 0.5, 1), nrow=2)
data <- mvrnorm(n, mu, Sigma)
print(head(data, 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
```

```r
sample_cov <- cov(data)
print(sample_cov)
```

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

```r
print(sample_cov - Sigma)
```

```
##             [,1]       [,2]
## [1,] 0.04437988 0.03921573
## [2,] 0.03921573 0.10450781
```

All of the sample variances and covariance are slightly higher than the true one, but not much. They are very similar to the each other.

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), nrow = n)
A <- chol(Sigma)
data <- t(A %*% t(Z)) + matrix(rep(mu, n), ncol = 2, byrow = TRUE)
head(data, 5)
```

```
##            [,1]      [,2]
## [1,] -0.0695286 1.2325719
## [2,]  0.2225159 0.3352784
## [3,]  0.9742218 3.4027020
## [4,]  2.8549158 2.4497009
## [5,]  1.3015828 1.9516325
```

```
sample_cov <- cov(data)
print(sample_cov)
```

```
##           [,1]      [,2]
## [1,] 1.3781020 0.4935851
## [2,] 0.4935851 0.8028422
```

```
print(sigma)
```

```
## function (object, ...)
## UseMethod("sigma")
## <bytecode: 0x1050296a8>
## <environment: namespace:stats>
```

```
print(sample_cov - Sigma)
```

```
##              [,1]         [,2]
## [1,]  0.378101986 -0.006414885
## [2,] -0.006414885 -0.197157820
```

In general, all of the sample variances and covariance are close the true one. But compared to (a), the difference has become larger.

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).

```
mymvnorm <- function(n, mu, sigma) {
Z <- matrix(rnorm(n * length(mu)), nrow = n)
A <- chol(sigma)
data <- t(A %*% t(Z)) + matrix(rep(mu, n), ncol = length(mu), byrow = TRUE)
return(data)
}
set.seed(1)
mu <- c(1, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), nrow = 2)
n <- 2000
data_function <- mymvnorm(n, mu, sigma)
head(data_function, 5)
```

```
##            [,1]      [,2]
## [1,] -0.0695286 1.2325719
## [2,]  0.2225159 0.3352784
## [3,]  0.9742218 3.4027020
## [4,]  2.8549158 2.4497009
## [5,]  1.3015828 1.9516325
```

```
set.seed(1)
mu <- c(1, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), nrow = 2)
n <- 2000
data_function <- mymvnorm(n, mu, sigma)
head(data_function, 5)
```

```
##            [,1]      [,2]
## [1,] -0.0695286 1.2325719
## [2,]  0.2225159 0.3352784
## [3,]  0.9742218 3.4027020
## [4,]  2.8549158 2.4497009
## [5,]  1.3015828 1.9516325
```

```
sample_cov <- cov(data_function)
print(sample_cov)
```

```
##           [,1]      [,2]
## [1,] 1.3781020 0.4935851
## [2,] 0.4935851 0.8028422
```

```
print(sigma)
```

```
##      [,1] [,2]
## [1,]  1.0  0.5
## [2,]  0.5  1.0
```

```r
print(sample_cov - sigma)
```

```
##              [,1]         [,2]
## [1,]  0.378101986 -0.006414885
## [2,] -0.006414885 -0.197157820
```

The result is same with question (b). In general, all of the sample variances and covariance are close the true one. But compared to (a), the difference has become larger.

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 use ChatGPT to analyze the questions and generate the draft R code. It helps me understand the logic behind generating multivariate normal random variables. And provide detail explanation for the R code it generated. The code it generates have several errors. For question b and c, the code it generated transformed matrix in a incorrect order and the dimension of the mu it generated is incorrect. I modified the code by changing the dimension of mu and the order of transforming the matrix.

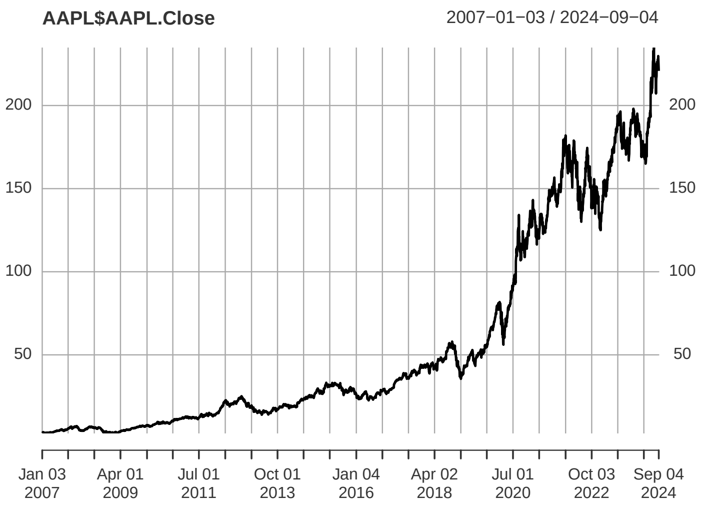
Question 2 (Data Manipulation Plots)

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).

```r
  library(quantmod)
  getSymbols("AAPL")
```

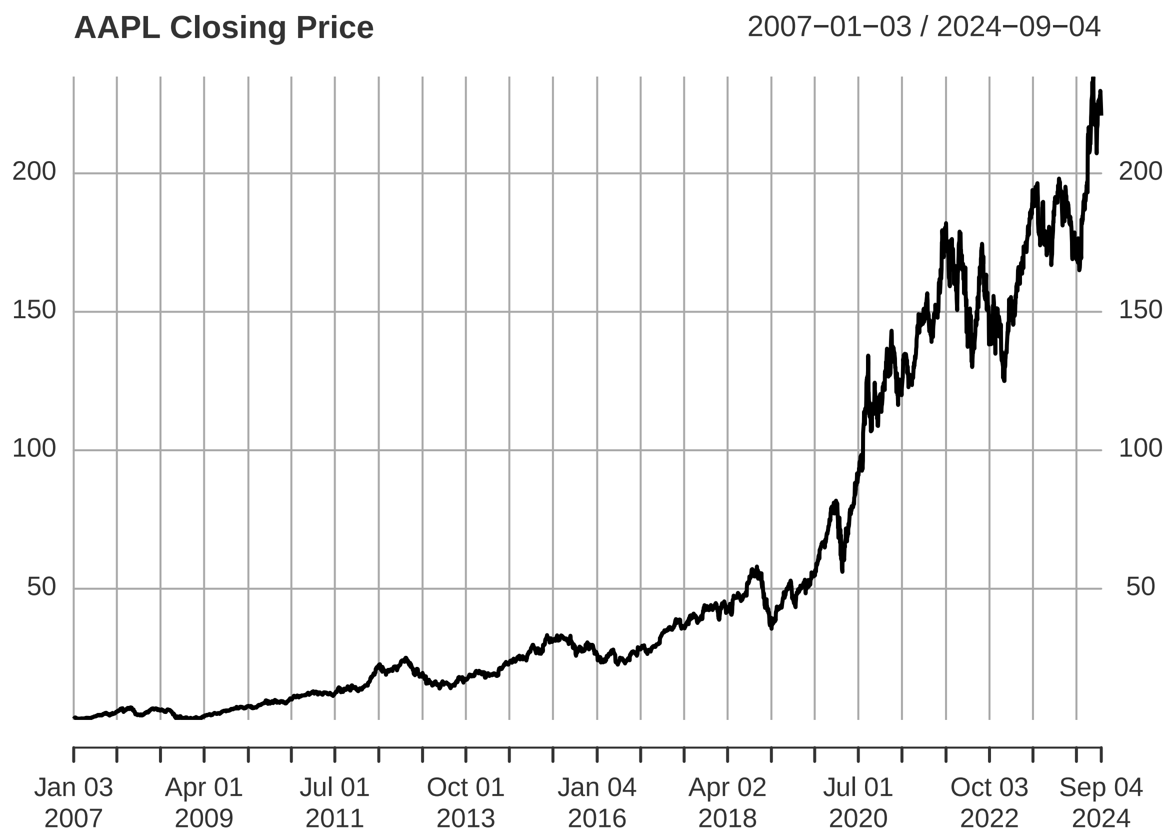
```
## [1] "AAPL"
```

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

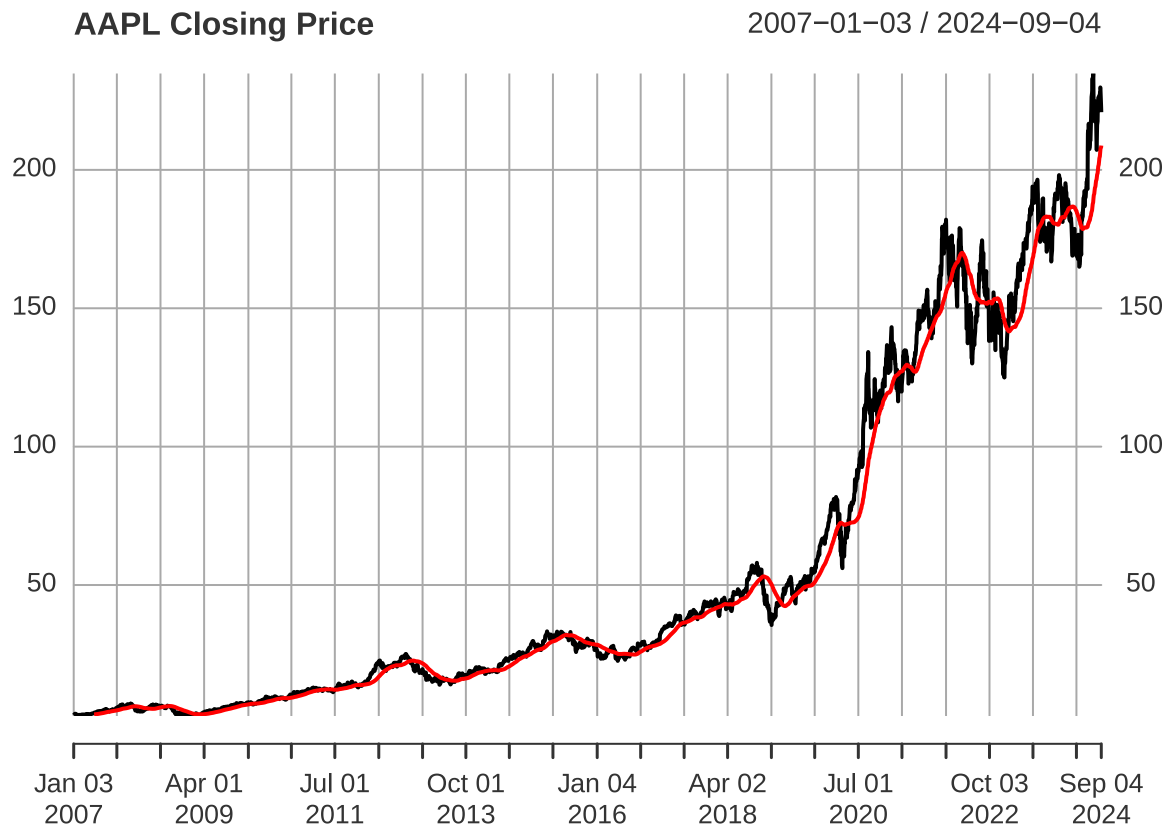


a. [20 points] Calculate a 90-day moving average of the closing price of `AAPL` and plot it on the same graph. Moving average means that for each day, you take the average of the previous 90 days. Please do this in two ways: 1) there is a built-in function called `SMA` in the `quantmod` package; 2) write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

```r
aapl_prices <- AAPL$AAPL.Close
plot(AAPL$AAPL.Close, pch = 19, main = "AAPL Closing Price", col = "black", type = "l")
```

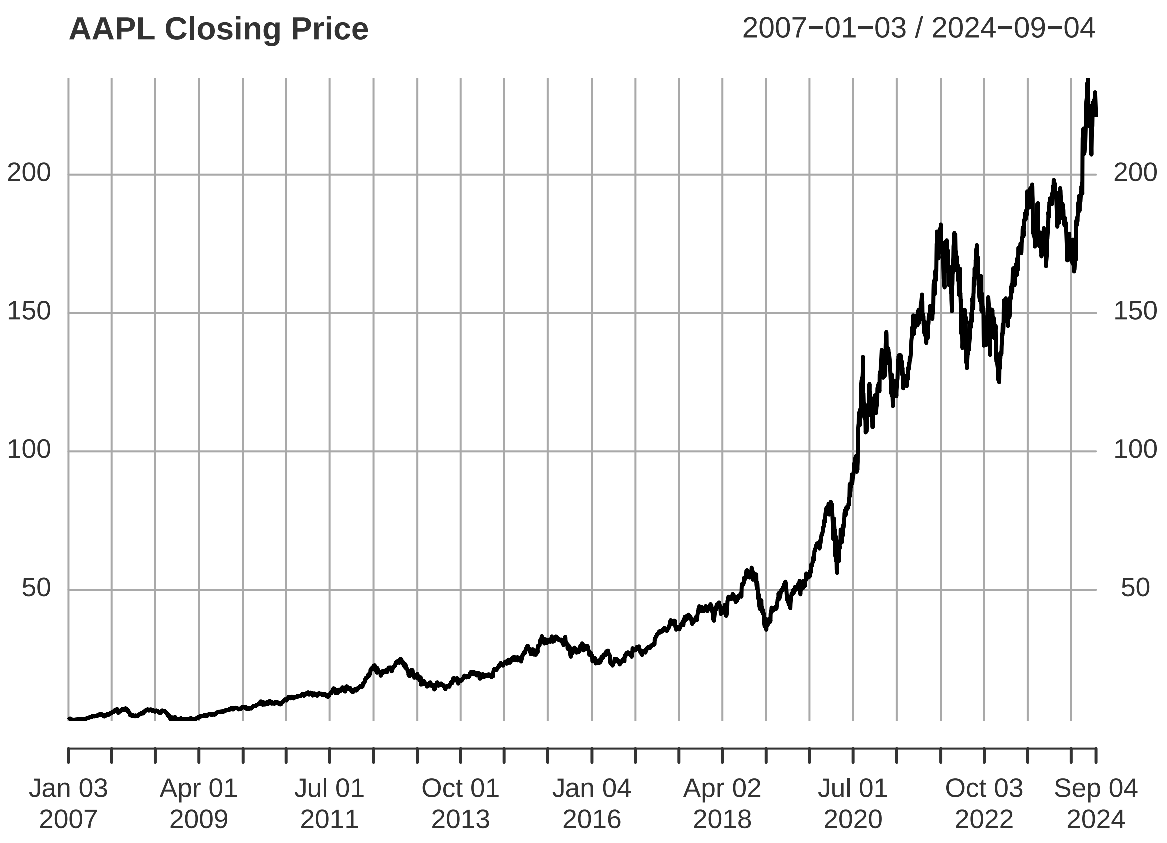


```r
AAPL_SMA_90 <- SMA(aapl_prices, n = 90)
lines(AAPL_SMA_90, col = "red", lwd = 2)
```

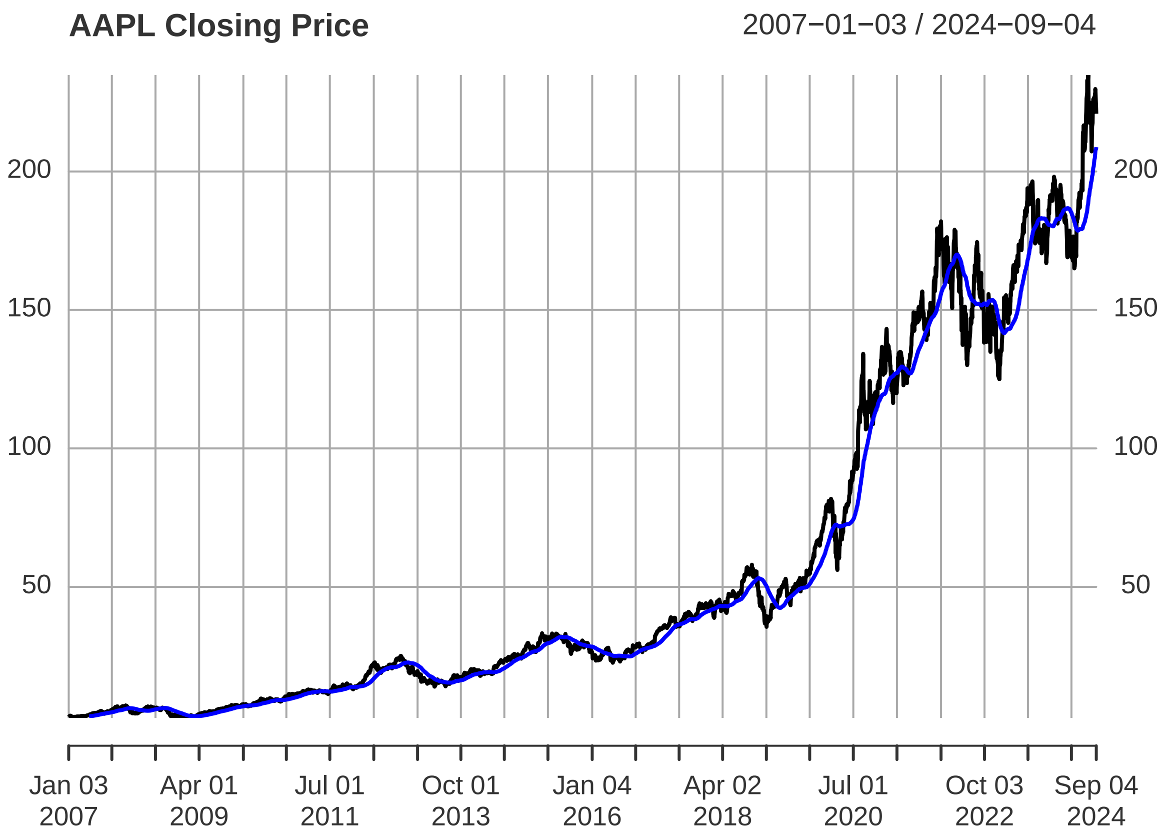


2) Custom function

```
plot(aapl_prices, pch = 19, main = "AAPL Closing Price", col = "black", type = "l")
```



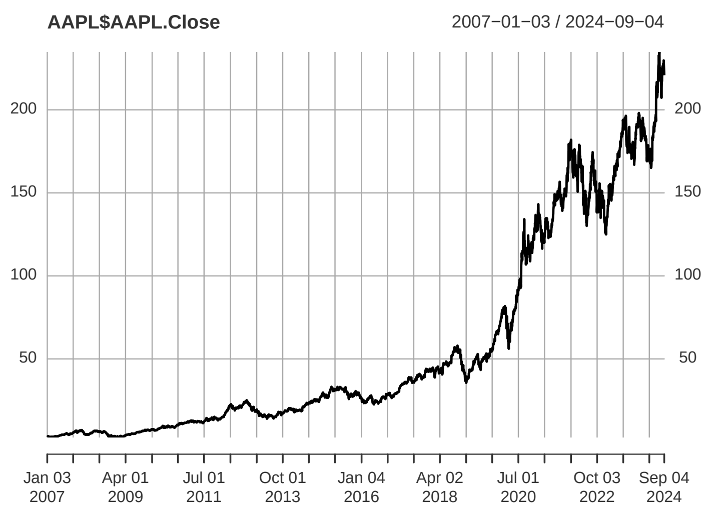
```r
calculate_moving_average <- function(prices, n) {
  ma <- rep(NA, length(prices))
  for (i in n:length(prices)) {
    ma[i] <- mean(prices[(i-n+1):i], na.rm = TRUE)
  }
  return(ma)
}
time_index <- index(Cl(AAPL))
AAPL_custom_SMA_90 <- xts(calculate_moving_average(Cl(AAPL), 90), order.by = time_index)
lines(AAPL_custom_SMA_90, col = "blue", lwd = 2)
```



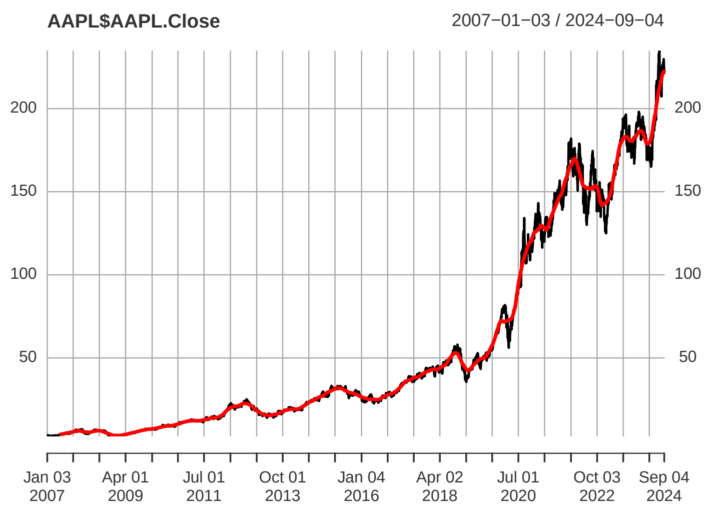
b. [15 points] I have an alternative way of writing this function.

```
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)
```



```r
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.

Difference:

- First chart (SMA): Uses a Simple Moving Average of the past 90 days to smooth the price, focusing on long-term trends and reducing noise. It is less sensitive to short-term changes and is better for long term analysis.

- Second chart: It use a 90-days window centered around the current point. It captures more short term fluctuations, following price movements more closely. It is more sensitive to short-term price movements, and is useful for short-term predictions.

Preference:

- The approach for the second chart may not be ideal because it relies on future data. But in real-world trading, future prices aren't available, making this method impractical for actual forecasting. We can't use future data to predict future prices so it's not suitable for real-time predictions.

- I prefer using the built-in SMA function(first chart). It's more efficient and reliable for most time series analysis. The function is optimized for time series operations, allowing for faster processing of large datasets. It eliminates the need to manually code the moving average logic and simplify implementation. And since it belongs to a well-maintained library, it is less possible to encounter bugs or unexpected issues.

Question 3 (Read/write Data)

a. [10 Points] The `ElemStatLearn` package [CRAN link] 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.

```
if (!require(devtools)) {
    install.packages("devtools")
}
```

```
## Loading required package: devtools

## Loading required package: usethis
```

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

```
## Using GitHub PAT from the git credential store.

## Skipping install of 'ElemStatLearn' from a github remote, the SHA1 (253e5401) has not changed since 
##   Use `force = TRUE` to force installation
```

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.

```
library(ElemStatLearn)
data(ozone, package = "ElemStatLearn")
head(ozone, 5)
```

```
##   ozone radiation temperature wind
## 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
```

```
write.csv(ozone, "ozone_data.csv", row.names = FALSE)
ozone_read <- read.csv("ozone_data.csv")
head(ozone_read, 5)
```

```
##   ozone radiation temperature wind
## 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
```