HW3_tdolkar

Due Wednesday Sep 30

Tsering Dolkar

9/24/2020

Problem 1

I did primer on Rstudio cloud.

Problem 2

Created the Rmd file

Problem 3

In the lecture, there were two links to programming style guides.

- Google's R style guide: https://google.github.io/styleguide/Rguide.xml

- Hadley Wickam's Style Guide: http://r-pkgs.had.co.nz/style.html

While the two styling had some differences, it mainly encouraged coders to write informative comments between lines of code, to choose informative and original variable/function names, and to make the code neat and readable. I felt that these were advices that will not only be helpful in understanding code while working on projects with other people, but also when I revisit my codes after some time has passed.

Problem 4

Good programming practices start with this homework. In the last homework, you imported, munged, cleaned and summarized datasets from Wu and Hamada's *Experiments: Planning, Design and Analysis.*

Problem 5

#dplyr + summarize + group_by

```r
summarise_data <- function(our_data){
  #function to calculate summary of a dataframe, returns a vector of the summary
  observer_summary <- double(length = 5)
  for(i in 1:5){
    if(i == 1)
```

```
      observer_summary[i] <- mean(our_data[,1])
    if(i == 2)
      observer_summary[i] <- mean(our_data[,2])
    if(i == 3)
      observer_summary[i] <- sd(our_data[,1])
    if(i == 4)
      observer_summary[i] <- sd(our_data[,2])
    if(i == 5)
      observer_summary[i] <- cor(our_data[,1], our_data[,2])
  }
  return(observer_summary)
}
```

```
# We'll find the summary for each observer:
# We will be returned a vector of length 5 for each of 13 observers from the function
# summarise_data such that
# 1. mean of dev 1
# 2. mean of dev 2
# 3. standard dev of dev 1
# 4. standard dev of dev 2
# 5. correlation between dev 1 and 2
# are the values respectively in a row.

#separate the data into vectors to make it less confusing to work with first
observer <- observations$Observer
dev1 <- observations$dev1
dev2 <- observations$dev2

#initialize vectors we will need:
dev1_by_observer <- double(0)
dev2_by_observer <- double(0)
summary_statistics <- data.frame()

#we have a nested for loop here. The outer for loop keeps track of observer 1 to 13 and
#the inside for loop looks for all the data by the said observer in the dataset from top to bottom
#once.
for(i in 1:13){
   track_observer <- i
   for(j in 1:length(observer)){
     if(track_observer == observer[j]){
     dev1_by_observer <- c(dev1_by_observer, dev1[j])
     dev2_by_observer <- c(dev2_by_observer, dev2[j])
     }
   }
   raw_data <- data.frame(dev1_by_observer, dev2_by_observer)
   colnames(raw_data) <- c("dev1", "dev2")
   summary_statistics_each_observer <- cbind(rep(i, 5), summarise_data(raw_data))
   summary_statistics <- rbind(summary_statistics, summary_statistics_each_observer)
}
summary_statistics <- data.frame(rep(c("Mean_dev1", "Mean_dev2", "SD_dev1", "SD_dev2",
                                        "Cor_dev1_dev2"), 13), summary_statistics)
colnames(summary_statistics) <- c('V1','Observer','V3')
summary_statistics <- summary_statistics %>%
                            spread(key = V1, value = V3)
# reorder by column name
summary_statistics <- summary_statistics[c("Observer", "Mean_dev1", "Mean_dev2",
                                                        "SD_dev1", "SD_dev2", "Cor_dev1_dev2")]
# summary_statistics <- kable(summary_statistics)
print(summary_statistics)
```

```
##    Observer Mean_dev1 Mean_dev2  SD_dev1  SD_dev2 Cor_dev1_dev2
## 1         1  54.26610  47.83472 16.76982 26.93974   -0.06412835
## 2         2  54.26741  47.83277 16.73988 26.89010   -0.06635717
## 3         3  54.26738  47.83442 16.72686 26.87172   -0.06701886
## 4         4  54.26636  47.83388 16.72164 26.86388   -0.06638213
## 5         5  54.26515  47.83507 16.71903 26.85814   -0.06517409
## 6         6  54.26453  47.83427 16.71698 26.85592   -0.06459745
## 7         7  54.26514  47.83444 16.71564 26.85435   -0.06515566
## 8         8  54.26548  47.83462 16.71463 26.85269   -0.06563368
## 9         9  54.26552  47.83427 16.71408 26.85168   -0.06596437
## 10       10  54.26570  47.83480 16.71366 26.85004   -0.06566406
## 11       11  54.26609  47.83500 16.71340 26.84937   -0.06600791
## 12       12  54.26616  47.83471 16.71318 26.84883   -0.06605520
## 13       13  54.26570  47.83510 16.71300 26.84777   -0.06601891
```

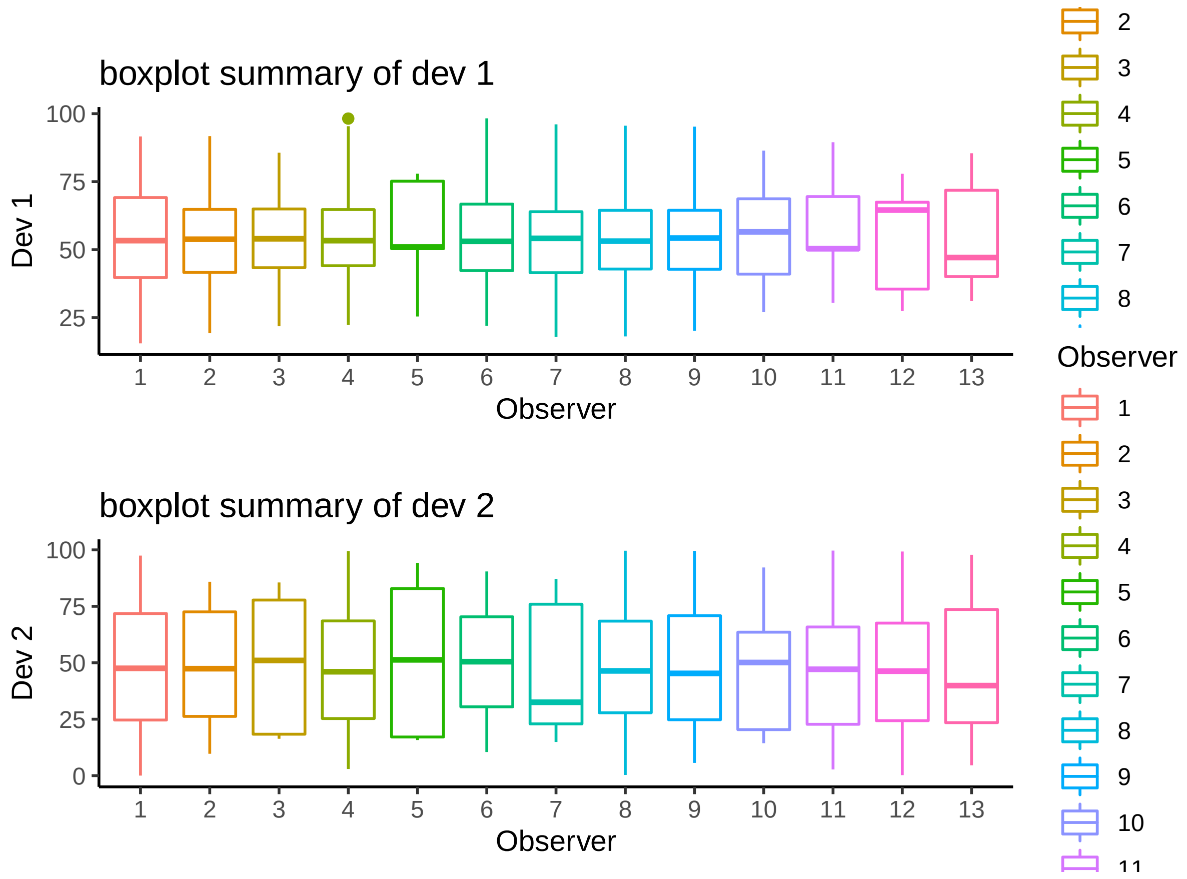
From this, it seems that device 2 has a lot bigger standard deviation than device 1 and device 1 and device 2 has negative and weak correlation.

```
observations$Observer <- as.factor(observations$Observer)

dev1 <- observations %>%
  ggplot(aes(x = Observer, y = dev1, colour = Observer)) +
  geom_boxplot() +
  labs(title = "boxplot summary of dev 1", x="Observer", y = "Dev 1") +
  theme_classic()

dev2 <- observations %>%
  ggplot(aes(x = Observer, y = dev2, colour = Observer)) +
  geom_boxplot() +
  labs(title = "boxplot summary of dev 2", x="Observer", y = "Dev 2") +
  theme_classic()

figure1 <- multi_panel_figure(columns = 1, rows = 2, panel_label_type = "none")
figure1 %<>%
  fill_panel(dev1, column = 1, row = 1) %<>%
  fill_panel(dev2, column = 1, row = 2)
figure1
```



From these plots, we can see that the deviation from median or the IQR is much bigger for device 2 than device 1 for almost all observers. It also seems that the median is close to the mean of the device. Perhaps the distribution for the two devices is normal distribution?

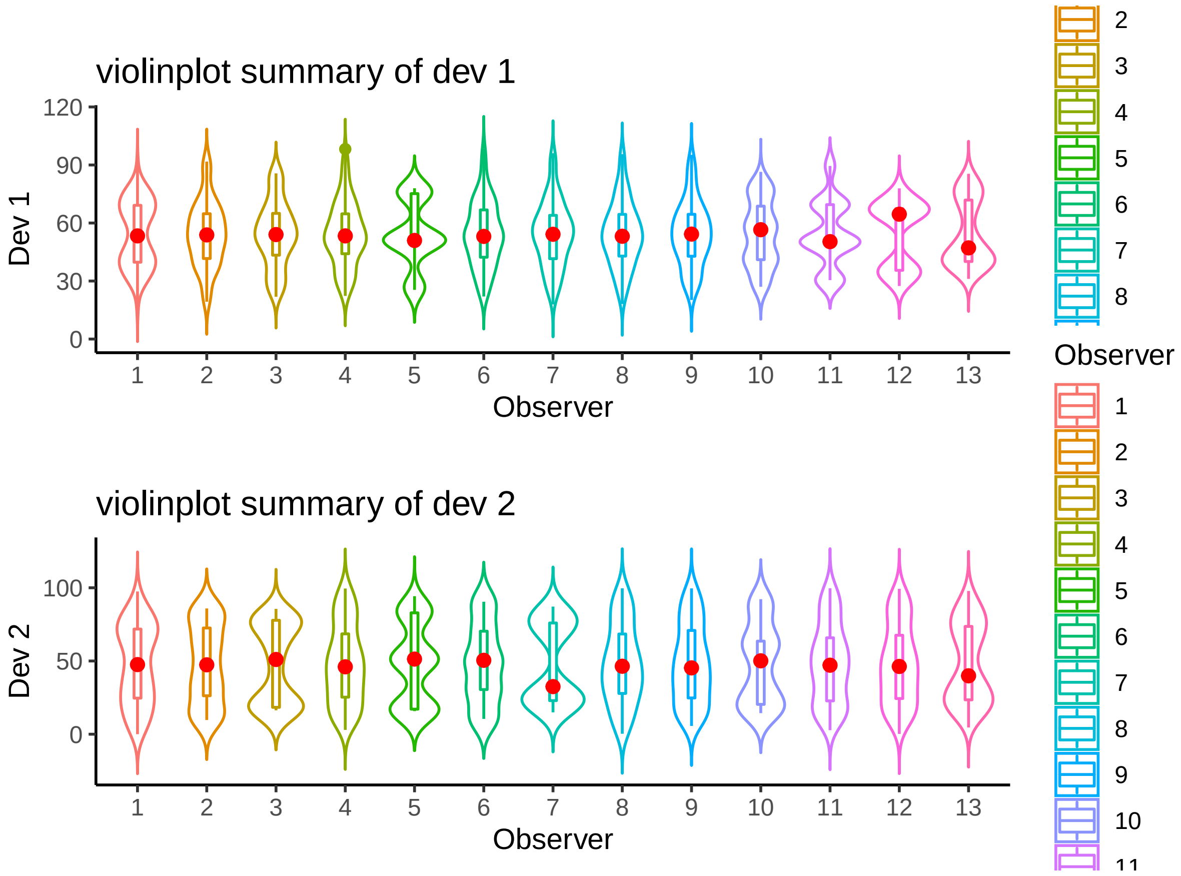
```r
dev1 <- observations %>%
  ggplot(aes(x = Observer, y = dev1, colour = Observer)) +
  geom_violin(trim = F) +
  geom_boxplot(width=0.1) +
  stat_summary(fun.y=median, geom="point", size=2, color="red") +
  labs(title = "violinplot summary of dev 1", x="Observer", y = "Dev 1") +
  theme_classic()
```

```
## Warning: ‘fun.y‘ is deprecated. Use ‘fun‘ instead.
```

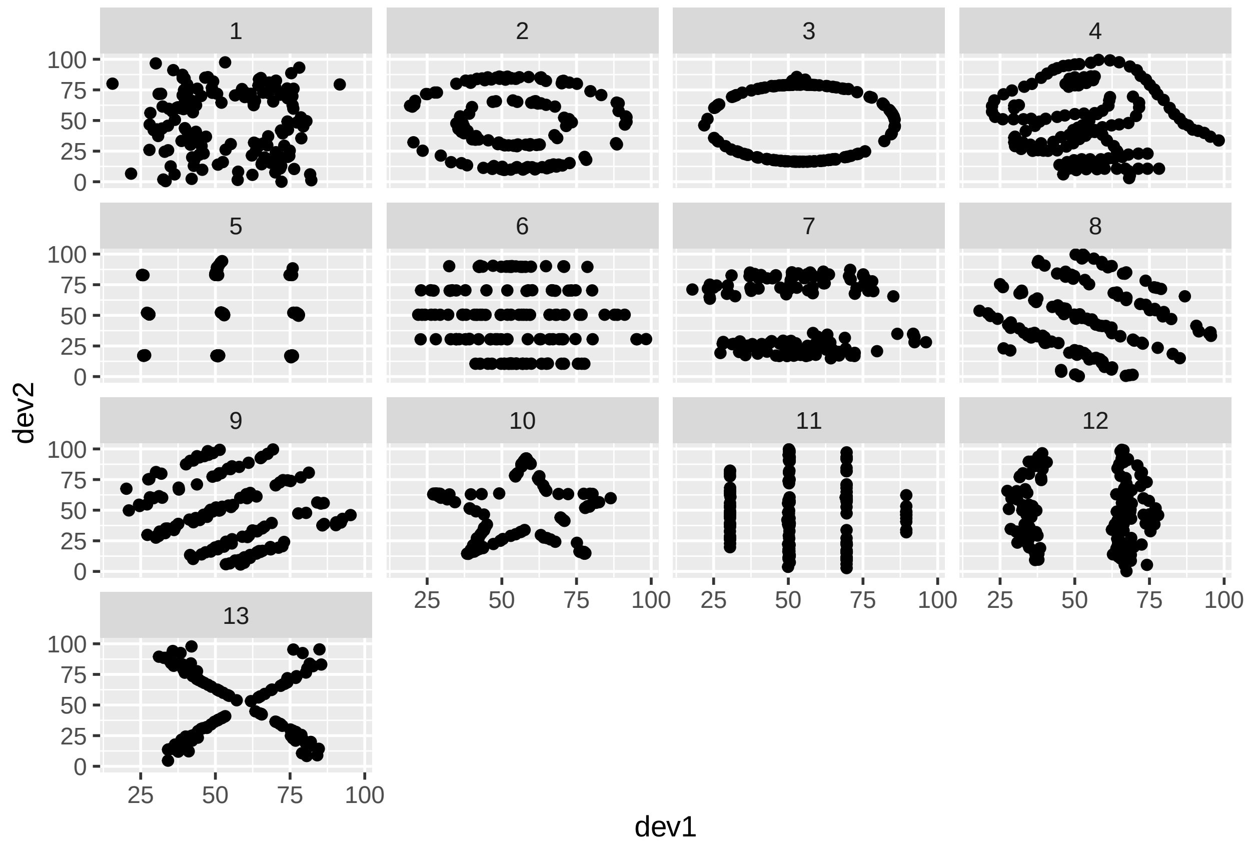
```r
dev2 <- observations %>%
  ggplot(aes(x = Observer, y = dev2, colour = Observer)) +
  geom_violin(trim = F) +
  geom_boxplot(width=0.1) +
  stat_summary(fun.y=median, geom="point", size=2, color="red") +
  labs(title = "violinplot summary of dev 2", x="Observer", y = "Dev 2") +
  theme_classic()
```

```
## Warning: ‘fun.y‘ is deprecated. Use ‘fun‘ instead.
```

```
figure1 <- multi_panel_figure(columns = 1, rows = 2, panel_label_type = "none")
figure1 %<>%
  fill_panel(dev1, column = 1, row = 1) %<>%
  fill_panel(dev2, column = 1, row = 2)
figure1
```



For the most part, it seems to give us the same information as we had in boxplot. The red dot on this plot is our mean. So therefore, we know that our median and mean is the same value. This implies dev1 and dev2 data is normal in distribution. One extra information from this plot is also that the shape of the individual violins displays frequencies of values.



Problem 6

```r
R_sum <- function(a, b, n){
#takes the interval(a,b) and partitions it in n
#pieces. Then returns the sum of all the area.
#the bigger the n value, the closer R_sum value
#gets to integral value.
  area_rectangle <- double()
  dx <- (b - a)/n
  for(i in 1:n){
    x <- a + dx*i
    f_x <- exp((-x^2)/2)
    area_rectangle[i] <- dx * f_x
  }
  return(sum(area_rectangle))
}
```

```r
how_many_parts <- seq(from = 600000, to = 1600000, by = 100000)
a <- 0
b <- 1
RS <- double()
track_RS <- data.frame()
report <- data.frame()
```

```r
integrand <- function(x) {exp((-x^2)/2)}
integral_value <- integrate(integrand, lower = 0, upper = 1)
integral_value <- integral_value$value

# calls the function and calculates the Riemann of sum for each
# increasing number of total partition.
for(i in how_many_parts){
  RS <- R_sum(a, b, i)
  slice_width <- ((b-a)/i)
  track_RS <- rbind(track_RS, c(slice_width, RS))
  # if riemann sum is within e^(-6) points from the integral value, store it in
  # dataframe report.
  if(abs(RS - integral_value) <= 1e-6){
  report <- rbind(report, c(slice_width, RS))
  }
}
colnames(track_RS) <- c("slice_width", "Riemann_Sum")
if(length(report) > 0){
  colnames(report) <- c("slice_width", "Riemann_Sum")
}
print(report)
```

```
##     slice_width Riemann_Sum
## 1  1.666667e-06   0.8556241
## 2  1.428571e-06   0.8556241
## 3  1.250000e-06   0.8556241
## 4  1.111111e-06   0.8556242
## 5  1.000000e-06   0.8556242
## 6  9.090909e-07   0.8556242
## 7  8.333333e-07   0.8556242
## 8  7.692308e-07   0.8556242
## 9  7.142857e-07   0.8556243
## 10 6.666667e-07   0.8556243
## 11 6.250000e-07   0.8556243
```

```r
summary(report)
```

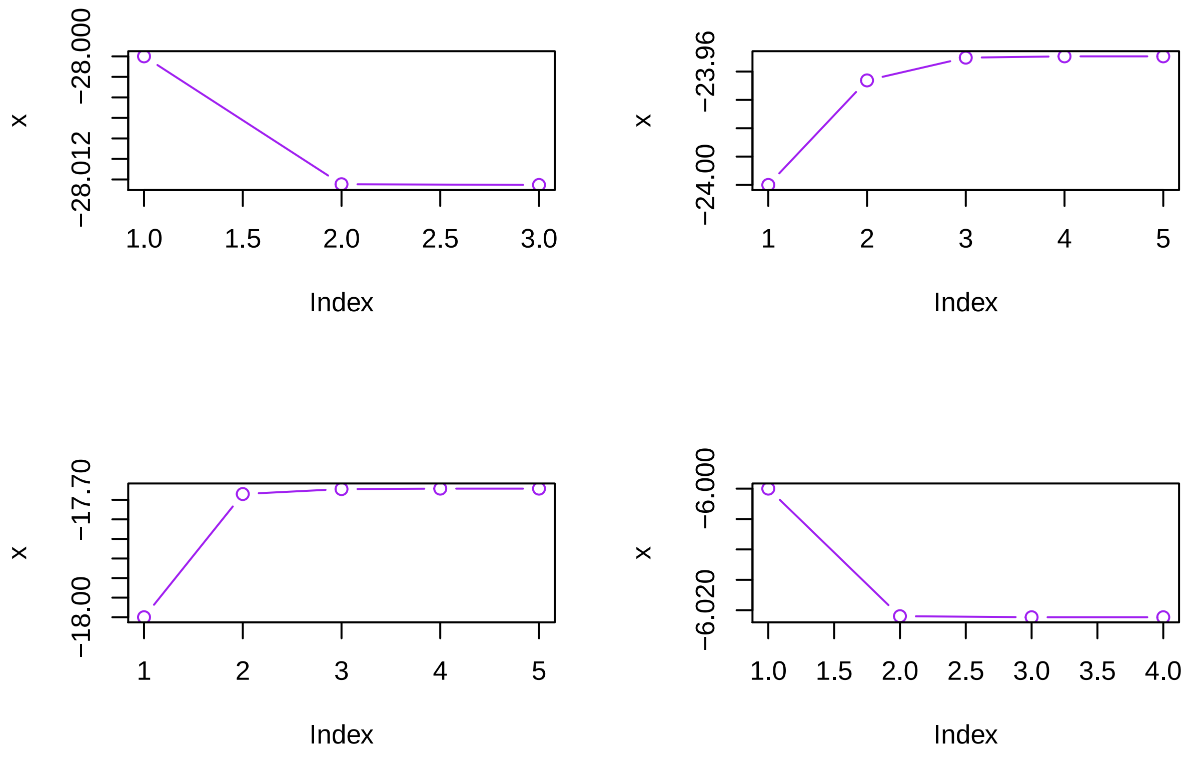
```
##   slice_width         Riemann_Sum
##  Min.   :6.250e-07   Min.   :0.8556
##  1st Qu.:7.418e-07   1st Qu.:0.8556
##  Median :9.091e-07   Median :0.8556
##  Mean   :9.976e-07   Mean   :0.8556
##  3rd Qu.:1.181e-06   3rd Qu.:0.8556
##  Max.   :1.667e-06   Max.   :0.8556
```

The width necessary to obtain an answer within $1e^{-}6$ of the analytical solution for me was the min value of slice_width. At that slice_width, I get the min value of Riemann_Sum. I could make the answer more accurate by lowering the by value in seq(how_many_parts)[for example, by = 1]. However, that slows down the compile time on my personal computer to hours.

Problem 7

```r
N_method <- function(x1 = -5){
  #function takes in a first approximation of the root x0 of the function 3^x-sin(x)+cos(5*x)
  #function uses Newton's method of approximate the root of the function
  #function terminates when successive estimates are within 1e-04 of each other
  seq <- 100
  x <- double(0)
  x[1] <-  x1
  # f <- 3^x- sin(x) + cos(5*x)
  # f_diff <- 3^x * log(3) - cos(x) - sin(5 * x) * 5
  for(i in 2:seq){
    if(3^x[i-1] * log(3) - cos(x[i-1]) - sin(5 * x[i-1]) * 5 == 0){
    stop("choose a new starting place")
    }
    x[i] <- x[i-1] - ((3^x[i-1]- sin(x[i-1]) + cos(5*x[i-1]))/
                          (3^x[i-1] * log(3) - cos(x[i-1]) - sin(5 * x[i-1]) * 5))
    if (abs(x[i]-x[i-1])<1e-04){
      break
    }
  }
  #For the initial estimate of:
  print(x[1])
  #Xn approaches 0 at:
  print(tail(x,1))
  return(plot(x, type = 'b', col = 'purple'))
}
```

```r
# we have:
#f <- expression(3^x- sin(x) + cos(5*x))
init_x <- c(-28, -24, -18, -6)
par(mfrow = c(2,2))
for(i in 1:length(init_x)){
  p = N_method(init_x[i])
}
```



```r
p
```

```
## NULL
```

Problem 8

```r
#Given simulated data from question
X <- cbind(rep(1,100),rep.int(1:10,time=10))
beta <- c(4,5)
y <- X%*%beta + rnorm(100)

# precalculation and vectors for SST using for_loop:
y_bar <- mean(y)
SST <- double(0)

# required values and matrices for SST using matrix operation:
n <- 100
J <- matrix(1, nrow = n, ncol = n)
I <- diag(x = 1, nrow = n, ncol = n)

times <- microbenchmark(SST1 <- {for(i in 1:length(y)){ SST[i] <- (y[i] - y_bar)^2};sum(SST)},
               SST2 <- {t(y)%*%(I - (1/n)*J)%*%y}, times = 100, unit = "ms")
print(times)
```

```
## Unit: milliseconds
##                                                                                                   expr
##  SST1 <- {     for (i in 1:length(y)) {         SST[i] <- (y[i] - y_bar)^2     }     sum(SST) }
##                                                   SST2 <- {     t(y) %*% (I - (1/n) * J) %*% y }
##       min        lq       mean    median        uq       max neval
##  2.545715 2.7593890 3.37196252 2.8960875 3.2057695 16.500412   100
##  0.024941 0.0277965 0.04361442 0.0409165 0.0495545  0.105255   100
```

```r
print(SST1)
```

```
## [1] 20429.84
```

```r
print(SST2)
```

```
##          [,1]
## [1,] 20429.84
```