

# HW3\_jaeyounglee

Jaeyoung Lee

September 26, 2020

Remember to adhere to both Reproducible Research and Good Programming Practices, ie describe what you are doing and comment/indent code where necessary.

## Problem 1

I learned with Primers

## Problem 2

New R markdown file

## Problem 3

In the lecture, there were two links to programming style guides. What is your takeaway from this and what specifically are *you* going to do to improve your coding style?

From *Google's R Style Guide*, I learned naming objects and functions. Also, I found that using `::` is really helpful. The symbol helps us to avoid risk for name collisions. From *Hadley Wickam's Style Guide*, I figured out how important the naming is. The guide says file names should be meaningful, variable names should be nouns, and functions names should be verbs. For reproducible research, it is important to be easily readable. To improve readability, we need to care about indentation, spacing, and so on.

## Problem 4

Good programming practices start with this homework

## Problem 5

We need to create a single function which takes as input a two column data frame and returns a vector containing

1. mean of column 1
2. mean of column 2
3. standard dev of column 1
4. standard dev of column 2
5. correlation between column 1 and 2

```
##### Problem 5 #####
# Define a function using base R function
find_stats <- function(x){
  # Find means, standards deviations and correlation of the columns from the data frame
  # The input x is a two column data frame
  mean_cols <- apply(x, 2, mean) # Mean of each column
  sd_cols <- apply(x, 2, sd) # Standard deviation of each column
  cor_cols <- cor(x[1], x[2]) # Correlation between first two columns
  # cor_mat <- cor(x) # Correlation matrix of the data (General version)

  # The vector of statistics
  data_stats <- c(mean_cols, sd_cols, cor_cols)
  names(data_stats) <- c(paste("mean", colnames(x)),
                        paste("sd", colnames(x)), "corr")

  return(data_stats)
}
```

Now, apply the defined function above to data and get summary statistics

```
# Load data
# Multiple repeated measurements from two devices (dev1 and dev2) by thirteen Observers.
devices <- readRDS('HW3_data.rds')

# Loop through the Observers collecting the summary statistics via function "find_stats"
summary_stats <- NULL
for (i in 1:max(devices$Observer)){
  # Find a part of the data frame by each Observer
  devices_part <- devices[which(devices$Observer == i),]

  # Find statistics of the data frame, and store the statistics in a single data frame
  summary_stats <- rbind(summary_stats, find_stats(devices_part[2:3]))
}
summary_stats <- data.frame(summary_stats)

# We can conduct same procedure above with tidyverse
summary_stats_tidyverse <- devices %>% group_by(Observer) %>%
  summarize(mean_dev1 = mean(dev1), mean_dev2 = mean(dev2),
            sd_dev1 = sd(dev1), sd_dev2 = sd(dev2),
            corr = cor(dev1, dev2), .groups = 'rowwise')
```

a. A single table of the means, sd, and correlation for each of the 13 Observers (*?kable*).

```
# Table of summary statistics
knitr::kable(summary_stats)           # With the defined function
```

mean.dev1	mean.dev2	sd.dev1	sd.dev2	corr
54.26610	47.83472	16.76983	26.93974	-0.0641284
54.26873	47.83082	16.76924	26.93573	-0.0685864
54.26732	47.83772	16.76001	26.93004	-0.0683434
54.26327	47.83225	16.76514	26.93540	-0.0644719
54.26030	47.83983	16.76774	26.93019	-0.0603414
54.26144	47.83025	16.76590	26.93988	-0.0617148
54.26881	47.83545	16.76670	26.94000	-0.0685042
54.26785	47.83590	16.76676	26.93610	-0.0689797
54.26588	47.83150	16.76885	26.93861	-0.0686092
54.26734	47.83955	16.76896	26.93027	-0.0629611
54.26993	47.83699	16.76996	26.93768	-0.0694456
54.26692	47.83160	16.77000	26.93790	-0.0665752
54.26015	47.83972	16.76996	26.93000	-0.0655833

```
knitr::kable(summary_stats_tidyverse) # With tidyverse
```

Observer	mean_dev1	mean_dev2	sd_dev1	sd_dev2	corr
1	54.26610	47.83472	16.76983	26.93974	-0.0641284
2	54.26873	47.83082	16.76924	26.93573	-0.0685864
3	54.26732	47.83772	16.76001	26.93004	-0.0683434
4	54.26327	47.83225	16.76514	26.93540	-0.0644719
5	54.26030	47.83983	16.76774	26.93019	-0.0603414
6	54.26144	47.83025	16.76590	26.93988	-0.0617148
7	54.26881	47.83545	16.76670	26.94000	-0.0685042
8	54.26785	47.83590	16.76676	26.93610	-0.0689797
9	54.26588	47.83150	16.76885	26.93861	-0.0686092
10	54.26734	47.83955	16.76896	26.93027	-0.0629611
11	54.26993	47.83699	16.76996	26.93768	-0.0694456
12	54.26692	47.83160	16.77000	26.93790	-0.0665752
13	54.26015	47.83972	16.76996	26.93000	-0.0655833

The result of tables are the same. From the tables, we can see that the descriptive statistics (sample mean, sample standard deviations, and correlations) are similar for all Observers. Therefore, we can conclude that the distributions of dev1 and dev2 are almost identically distributed regardless of Observers.

Focusing on the statistics, the means of dev1 and dev2 seem to be different. The means of dev1 are larger than those of dev2. In contrast, the standard deviations of dev1 are smaller than those of dev2. From the correlations, we can notice that dev1 and dev2 have weak negative linear relationship.

However, to know whether the means and standard deviations are significantly different, we need some inferences for means and standard deviations.

For the rest of parts, I will use `summary_stats` data frame.

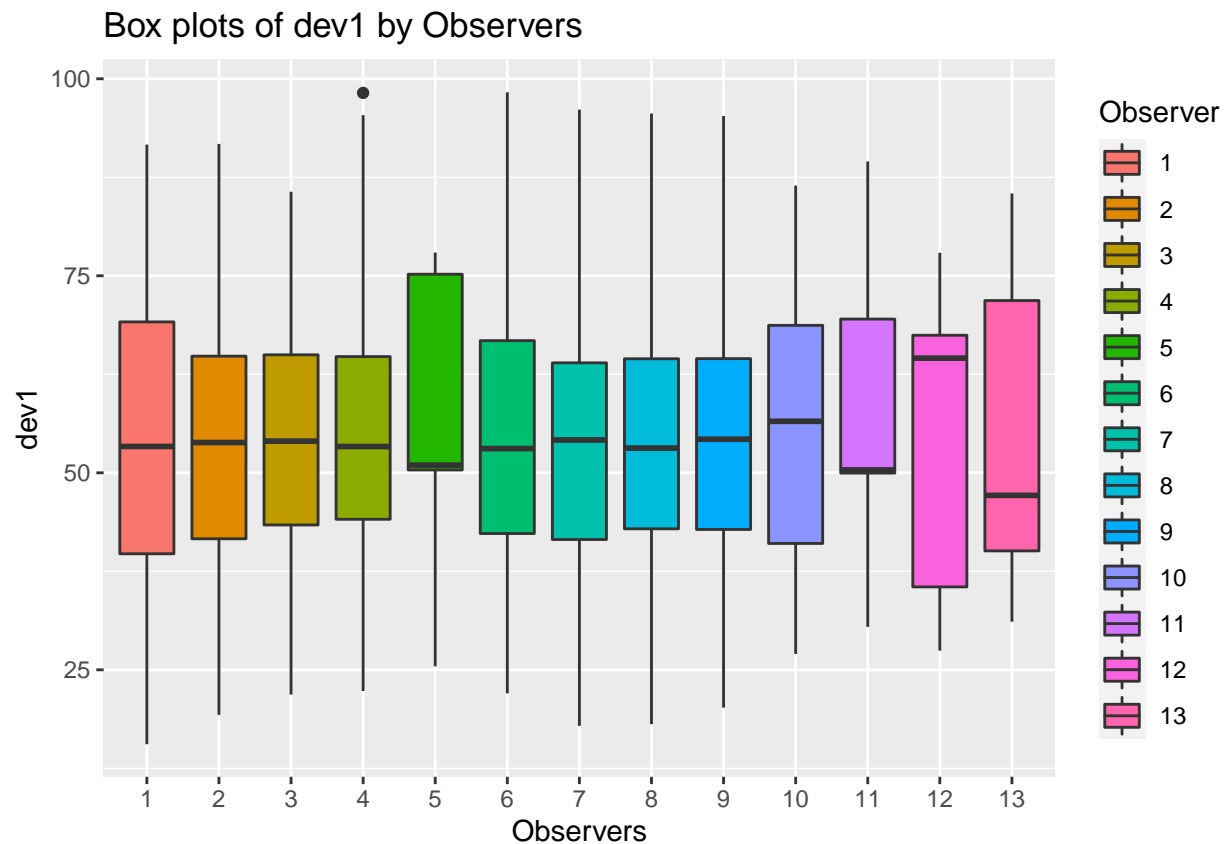
b. Box plots of dev1 and dev2s, by Observer.

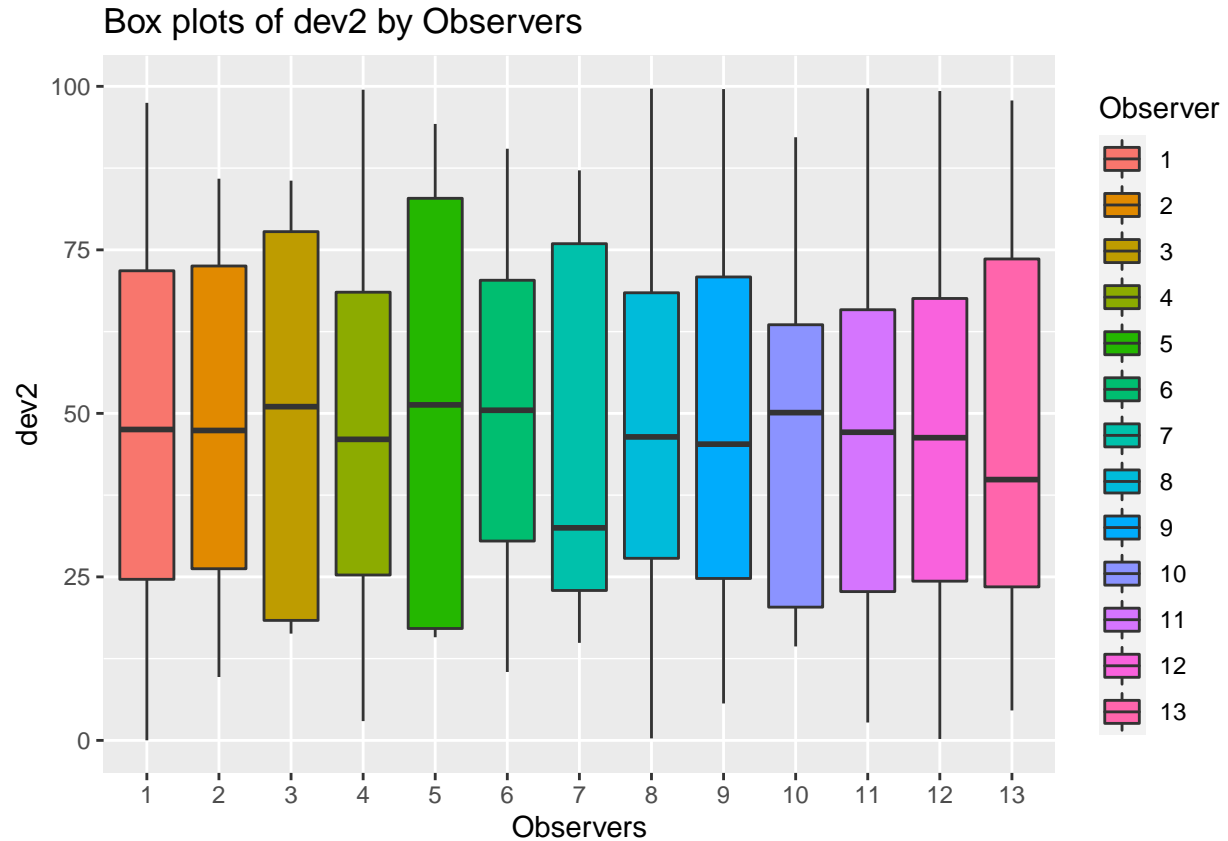
```
# Observers are categorical data
devices$Observer <- devices$Observer %>% factor()

# Box plots of dev1 by Observer
boxplot_dev1 <- ggplot(devices, aes(x=Observer, y= dev1, fill = Observer)) +
  geom_boxplot() +
  labs(title="Box plots of dev1 by Observers",x="Observers", y = "dev1")

# Box plots of dev2 by Observer
boxplot_dev2 <- ggplot(devices, aes(x=Observer, y= dev2, fill = Observer)) +
  geom_boxplot() +
  labs(title="Box plots of dev2 by Observers",x="Observers", y = "dev2")

boxplot_dev1; boxplot_dev2
```





The above plots are box plots of dev1 and dev2. The boxes are made of quartiles ( $Q_1$ ,  $Q_2$ , and  $Q_3$ ). The upper and bottom edges of the boxes are  $Q_1$ , and  $Q_3$  each. The line in the middle of the boxes is median  $Q_2$ . The upper and bottom ends of the whiskers represent the largest or smallest observations within 1.5 IQR from the quartiles. The black circle of Observer 4 from the dev1 box plot represents the outlier of the data.

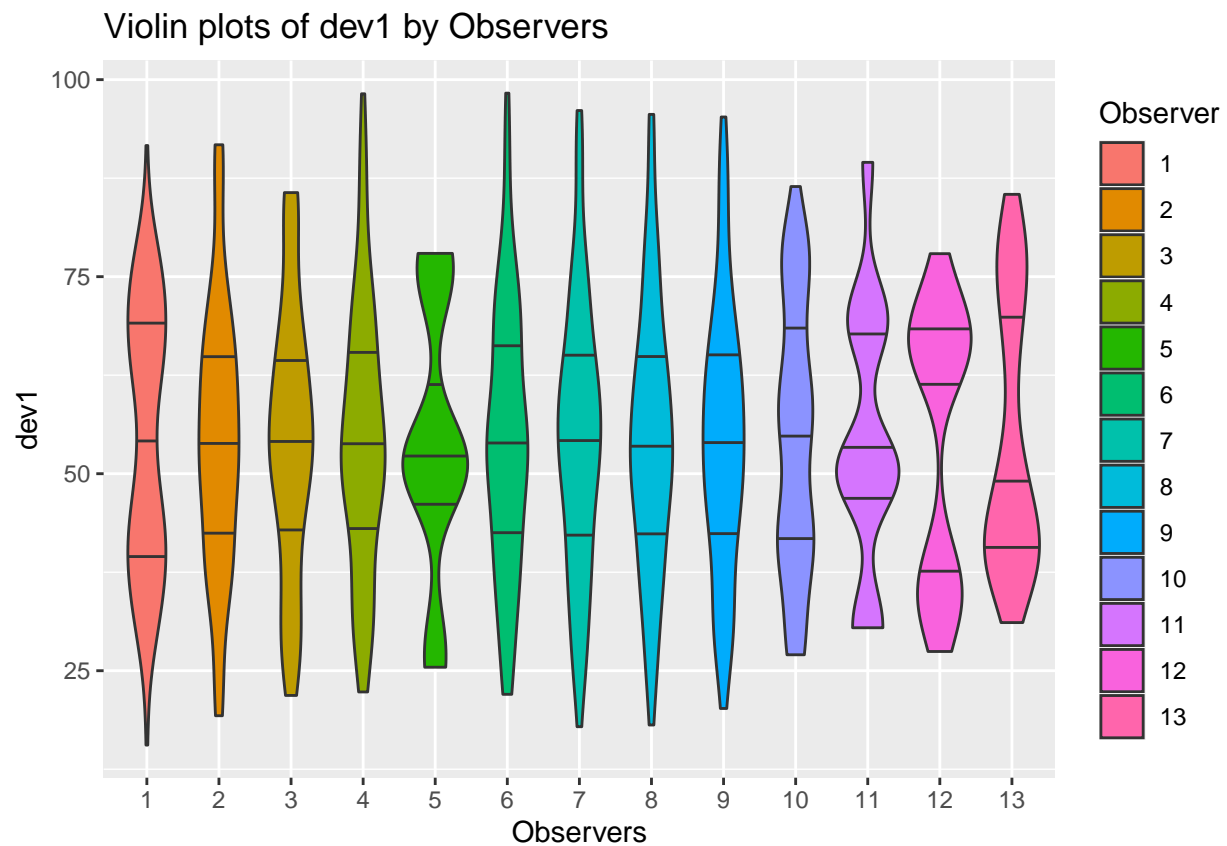
From the plots, we can notice that the overall dispersion of dev1 is smaller than that of dev2. Also, we can see the distributions of dev1 and dev2 more clearly than just looking at the summary statistics.

c. Violin plots of dev1 and dev2 by Observer.

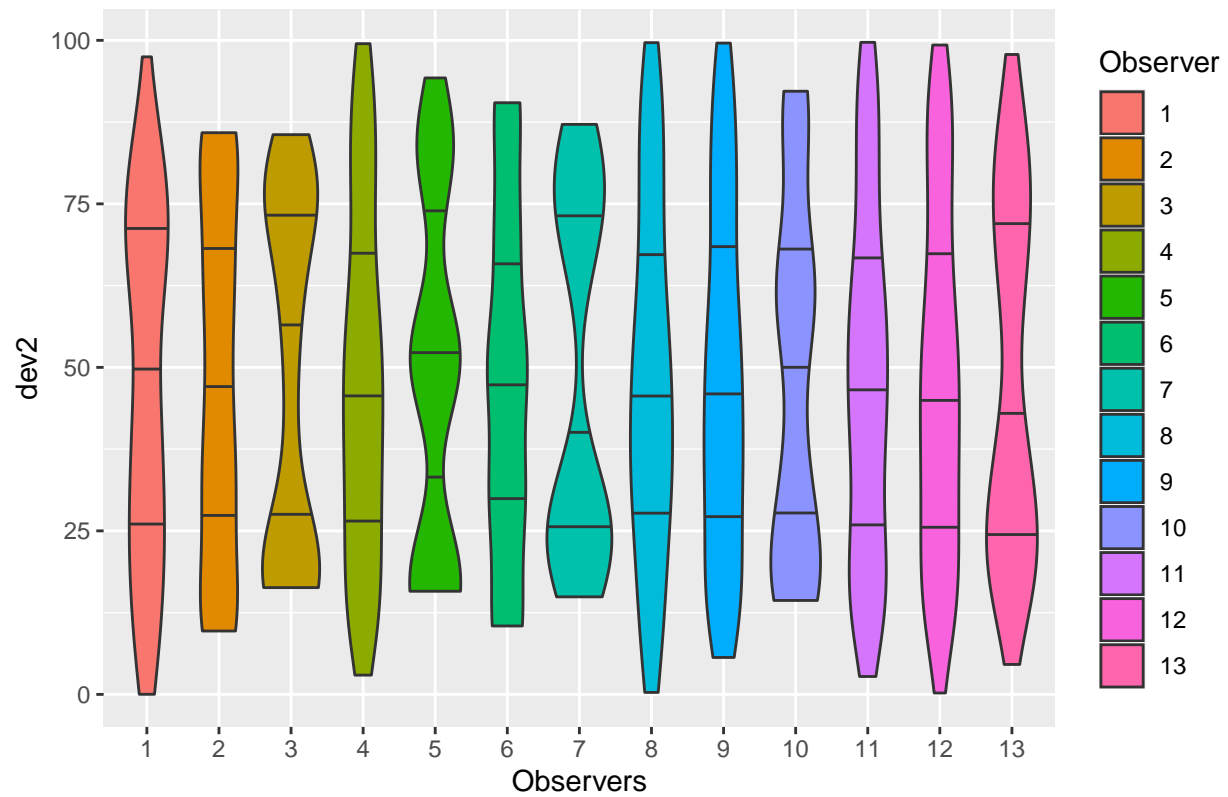
```
# Violin plots of dev1 by Observer
violin_dev1 <- ggplot(devices, aes(x=Observer, y= dev1, fill = Observer)) +
  geom_violin(draw_quantiles = c(0.25, 0.5, 0.75)) +
  labs(title="Violin plots of dev1 by Observers",x="Observers", y = "dev1")

# Violin plots of dev2 by Observer
violin_dev2 <- ggplot(devices, aes(x=Observer, y= dev2, fill = Observer)) +
  geom_violin(draw_quantiles = c(0.25, 0.5, 0.75)) +
  labs(title="Violin plots of dev2 by Observers",x="Observers", y = "dev2")

violin_dev1; violin_dev2
```



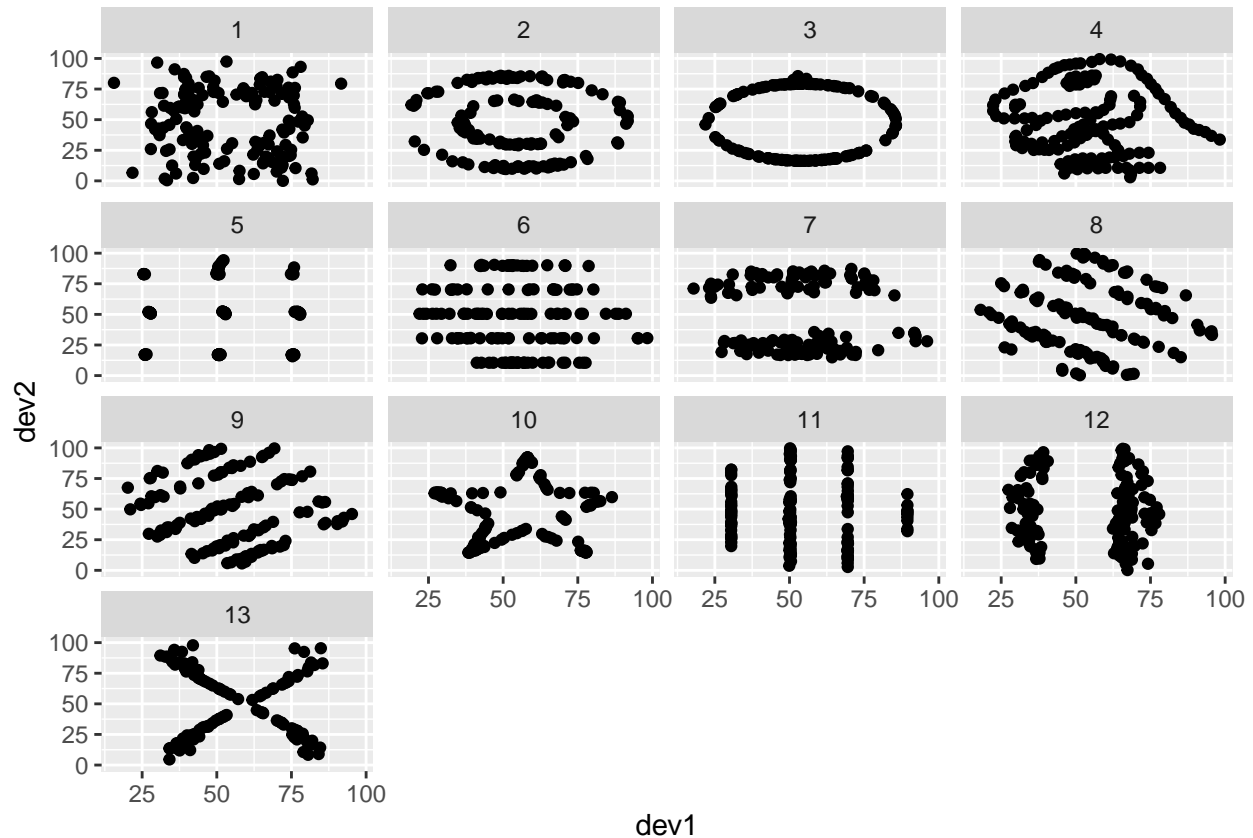
Violin plots of dev2 by Observers



Violin plots are alternatives of box plots. The lines of each violin are quartiles. The top line is  $Q_3$ , middle one is median, and the bottom is  $Q_1$ . Compared to box plots and summary statistics, violin plots have more information than box plots and summary statistics. Not only they show quartiles, but also show the distributions of each Observer and device. We can notice the shape of each probability density by the violin plots.

d. a scatter plot of the data using `ggplot`, `geom_point`, and add `facet_wrap` on `Observer`. For instance: `ggplot(df, aes(x=dev1,y=dev2)) + geom_point() + facet_wrap(Observer~.)`

```
# Scatter plots
ggplot(devices, aes(x=dev1,y=dev2)) + geom_point() + facet_wrap(Observer~.)
```



Pretty amazing. It made me laugh because I did not expect these pictures. I learned something from the scatter plots. When we conduct some “Exploratory Data Analysis”, it is not enough to show summary statistics. Also, it is better to use various kinds of plots to look at data. As a takeaway, using many different plots to get more information is not redundant. It is rather useful to avoid embarrassment from making erroneous conclusions.



## Problem 6

Some numerical methods are perfect candidates for functions. Create a function that uses Riemann sums to approximate the integral:

$$f(x) = \int_0^1 e^{-\frac{x^2}{2}} dx$$

The function should include as an argument the width of the slices used. Now use a looping construct (for or while) to loop through possible slice widths. Report the various slice widths used, the sum calculated, and the slice width necessary to obtain an answer within  $1e^{-6}$  of the analytical solution.

Define a function to be integrated. Also, define a function to find Riemann sum of a function in general. By choosing `method` argument, we can apply left, right, or midpoint Riemann sum.

```
##### Problem 6 #####
# Need to find \int_0^1 e^{-\frac{x^2}{2}} dx
# Let f(x) = e^{-\frac{x^2}{2}}
f <- function(x){
  value <- exp(-(x^2/2))
  return(value)
}

# Define a function to find Riemann sum
riemann_sum <- function(f, x, width, method = 'left'){
  # The input f is a function to be integrated
  # The input x is an integration interval
  # The input width is a width of each rectangle
  # The input method is to choose Riemann sum among the left, right or midpoint sum
  if(method == 'left'){
    # Left Riemann sum
    support <- seq(min(x),max(x), by = width) # Support of x
    fx <- f(support[1:(length(support)-1)]) # Height of rectangles
    integral <- width*sum(fx) # Left Riemann sum

  }else if(method == 'right'){
    # Right Riemann sum
    support <- seq(min(x),max(x), by = width) # Support of x
    fx <- f(support[2:length(support)]) # Height of rectangles
    integral <- width*sum(fx) # Right Riemann sum

  }else if(method == 'midpoint'){
    support <- seq(min(x),max(x), by = width) # Support of x
    support <- support - width/2 # To use midpoint as a domain
    fx <- f(support[2:length(support)]) # Height of rectangles
    integral <- width*sum(fx) # Right Riemann sum

  }else{
    integral <- c('Not vaild method')
  }

  return(integral)
}
```

Now, apply the function to calculate integrals using different widths.

```

# Need to find  $\int_0^1 e^{-\frac{x^2}{2}} dx$ 
# Apply different width
rectangle_width = c(.1, .01, .001, .0001)

# Apply various Riemann sum which are left, right and midpoint sums
x <- c(0,1) # Integration interval

result <- c('Left_sum', 'Midpoint_sum', 'Right_sum')
for(i in 1:4){
  result_riemann <-
    c(riemann_sum(f, x, width = rectangle_width[i], method = 'left'),
      riemann_sum(f, x, width = rectangle_width[i], method = 'midpoint'),
      riemann_sum(f, x, width = rectangle_width[i], method = 'right'))
  result <- data.frame(result, result_riemann)
}
names(result) <- c('Method', 'Width_0.1', 'Width_0.01', 'Width_0.001', 'Width_0.0001')
result

```

```

##           Method Width_0.1 Width_0.01 Width_0.001 Width_0.0001
## 1      Left_sum 0.8747922  0.8575867  0.8558211  0.8556441
## 2 Midpoint_sum 0.8558773  0.8556269  0.8556244  0.8556244
## 3      Right_sum 0.8354453  0.8536520  0.8554276  0.8556047

```

The value with calculator is 0.855624. We can see that the results of the `riemann_sum` function is almost same to the our expected result. Also, as the width of rectangles become smaller, the `Riemann_sum` values become closer to 0.855624. Therefore, we can conclude that the defined `riemann_sum` function works well.

## Problem 7

$$f(x) = 3^x - \sin(x) + \cos(5x) \quad (1)$$

Using Newton's method, find roots of the function above

```
##### Problem 7 #####

# f(x) = 3^x - sin(x) + cos(5*x)
f <- function(x){
  value <- 3^x - sin(x) + cos(5*x)
  return(value)
}

# f'(x) = 3^x*log(3) - cos(x) - 5*sin(5*x)
f_prime <- function(x){
  value <- 3^x*log(3) - cos(x) - 5*sin(5*x)
  return(value)
}

# Define the function to run Newton's method
find_sol_newton <- function(x, tolerance){
  # Input x is the initial value to begin the algorithm, t is tolerance
  # Initial values
  x_new <- x      # Initial value to operate Newton's method
  x_old <- 10000  # Initial value to operate while loop
  no_iter <- 1    # Number of iteration of the loop
  matrix_x <- x   # The matrix of iterated x values to trace the history

  # Newton's Method
  # When x is a vector, break the loop when FALSE for all values in x
  while(any(abs(x_new-x_old) > tolerance)){
    x_old <- x_new          # Update new x value
    x_new <- x_old - f(x_old)/f_prime(x_old) # Newton's method formula
    matrix_x <- cbind(matrix_x, x_new)      # Store all history of x values
    no_iter <- no_iter + 1                  # Count the number of iteration
  }

  history_x <- data.frame(t(matrix_x), 1:no_iter) # Stored x history

  return(list(Solution = x_new, History_of_x = history_x)) # Roots and histories
}

# Multiple initial values and tolerance
x0 <- -20:20
tolerance <- .0001

# Find roots and history of them using the defined function above
newton_sol <- find_sol_newton(x0,tolerance)

# History of finding roots
# Trace plot of finding solution
# Trace plot of x with the smallest initial value
history_x <- newton_sol$History_of_x          # Recorded history
```

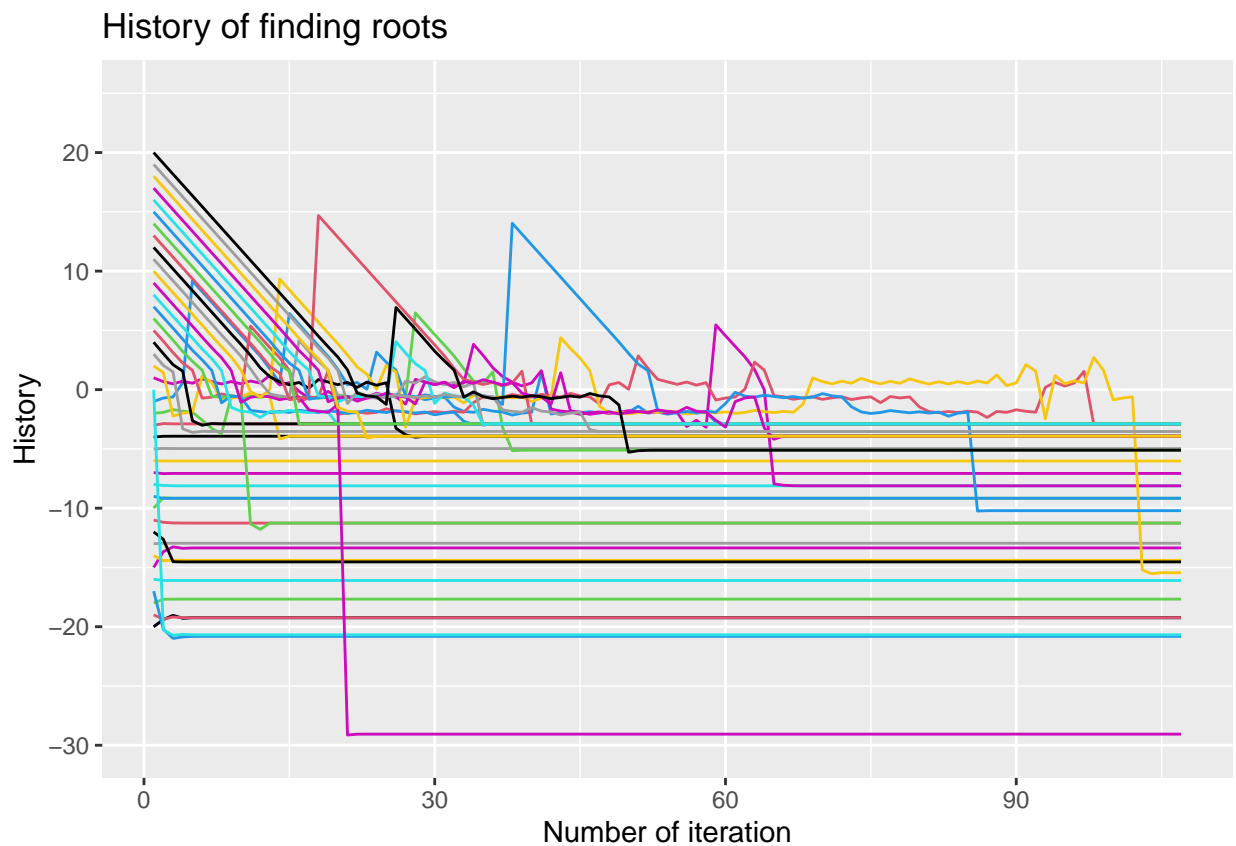
```

traceplot <- ggplot(data = history_x) +
  geom_line(aes(x = history_x[,42], y= history_x[,1]), col = 'cornflowerblue') +
  ylim(-30,25) +
  labs(title = 'History of finding roots',
        x = 'Number of iteration', y = 'History')

# Add trace plots of x with the other initial values
for (i in 1:41){
  traceplot <- traceplot + geom_line(aes_string(x = history_x[,42], y= history_x[,i]), col = i)
}

# History of finding roots
traceplot

```



```

# Round roots at 4th decimal place and delete replicated solutions
print("The roots"); newton_sol$Solution %>% round(digits = 4) %>% unique()

```

```
## [1] "The roots"
```

```
## [1] -19.2423 -17.6715 -20.8131 -16.1007 -13.3518 -14.3990 -12.9591 -14.5299
## [9] -11.2574 -9.1630 -8.1159 -7.0685 -6.0212 -4.9715 -3.9301 -2.8871
## [17] -5.1074 -20.6822 -3.5287 -29.0597 -15.4462 -10.2102
```

The results above are trace plots and final roots. The plots show that how x values have changed and converged to solutions. The number of initial values are 40, but some of them converged into the same root, so the number of roots are less than initial values.

## Problem 8

We want to calculate SST =

$$\sum_{i=1}^{100} (y_i - \bar{y})^2$$

The functions below calculate SST using:

- accumulating values in a for loop
- matrix operations only

```
##### Problem 8 #####
# Define functions to calculate SST
# Calculate SST using a for loop
find_sst_loop <- function(y){
  # The input y is a vector
  sst_element <- numeric(length(y))      # A vector of 0
  for (i in 1:length(y)){
    sst_element[i] <- (y[i] - mean(y))^2  # (Y_i - Y-bar)^2
  }
  sst <- sum(sst_element)                 # SST = sum of (Y_i - Y-bar)^2
  return(sst)
}

# Calculate SST using matrix operations only
find_sst_matrix <- function(y){
  # The input y is a vector
  J <- matrix(1, nrow = length(y), ncol = length(y)) # A matrix of 1, J
  sst <- t(y)%*(diag(length(y)) - J/length(y))%*%y  # SST with matrix notation
  return(sst)
}
```

We defined the functions above to find SST in different ways. One is using for loop and the other one only uses matrix operation. Now, apply these functions to data.

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

# SST of the functions
find_sst_loop(y)
```

```
## [1] 21025.42
```

```
find_sst_matrix(y)
```

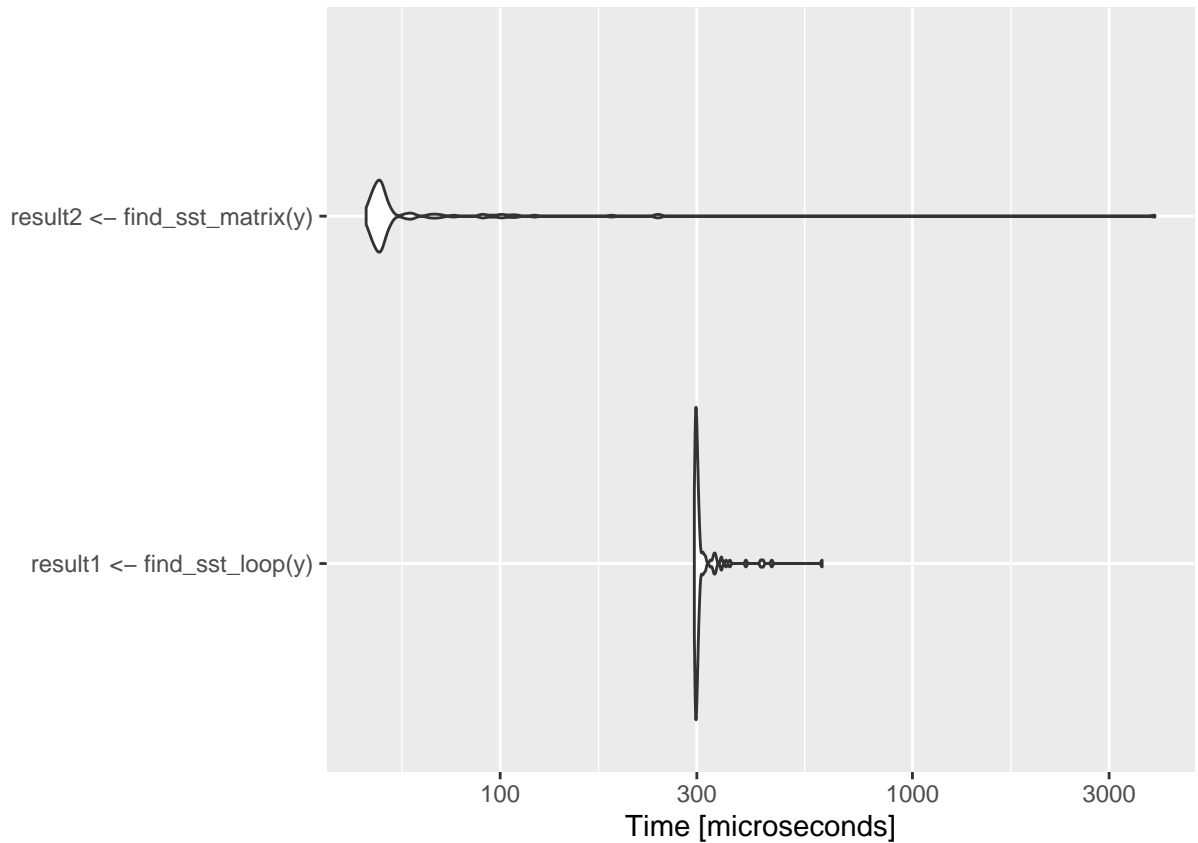
```
##           [,1]
## [1,] 21025.42
```

```

# Compare operation time using microbenchmark
times <- microbenchmark(result1 <- find_sst_loop(y),
                        result2 <- find_sst_matrix(y),
                        times = 100)
#identical(result1, result2)
autoplot(times)

```

## Coordinate system already present. Adding new coordinate system, which will replace the existing one



From the result above, we can know that the functions have the same result, but the matrix operation is much faster than the function using a for loop.

```
##### Problem 5 #####
# Define a function using base R function
find_stats <- function(x){
  # Find means, standards deviations and correlation of the columns from the data frame
  # The input x is a two column data frame
  mean_cols <- apply(x, 2, mean) # Mean of each column
  sd_cols <- apply(x, 2, sd) # Standard deviation of each column
  cor_cols <- cor(x[1], x[2]) # Correlation between first two columns
  # cor_mat <- cor(x) # Correlation matrix of the data (General version)

  # The vector of statistics
  data_stats <- c(mean_cols, sd_cols, cor_cols)
  names(data_stats) <- c(paste("mean", colnames(x)),
                        paste("sd", colnames(x)), "corr")

  return(data_stats)
}

# Load data
# Multiple repeated measurements from two devices (dev1 and dev2) by thirteen Observers.
devices <- readRDS('HW3_data.rds')

# Loop through the Observers collecting the summary statistics via function "find_stats"
summary_stats <- NULL
for (i in 1:max(devices$Observer)){
  # Find a part of the data frame by each Observer
  devices_part <- devices[which(devices$Observer == i),]

  # Find statistics of the data frame, and store the statistics in a single data frame
  summary_stats <- rbind(summary_stats, find_stats(devices_part[2:3]))
}
summary_stats <- data.frame(summary_stats)

# We can conduct same procedure above with tidyverse
summary_stats_tidyverse <- devices %>% group_by(Observer) %>%
  summarize(mean_dev1 = mean(dev1), mean_dev2 = mean(dev2),
            sd_dev1 = sd(dev1), sd_dev2 = sd(dev2),
            corr = cor(dev1, dev2), .groups = 'rowwise')

# Table of summary statistics
knitr::kable(summary_stats) # With the defined function
knitr::kable(summary_stats_tidyverse) # With tidyverse

# Observers are categorical data
devices$Observer <- devices$Observer %>% factor()

# Box plots of dev1 by Observer
boxplot_dev1 <- ggplot(devices, aes(x=Observer, y= dev1, fill = Observer)) +
  geom_boxplot() +
  labs(title="Box plots of dev1 by Observers", x="Observers", y = "dev1")

# Box plots of dev2 by Observer
```

```

boxplot_dev2 <- ggplot(devices, aes(x=Observer, y= dev2, fill = Observer)) +
  geom_boxplot() +
  labs(title="Box plots of dev2 by Observers",x="Observers", y = "dev2")

boxplot_dev1; boxplot_dev2

# Violin plots of dev1 by Observer
violin_dev1 <- ggplot(devices, aes(x=Observer, y= dev1, fill = Observer)) +
  geom_violin(draw_quantiles = c(0.25, 0.5, 0.75)) +
  labs(title="Violin plots of dev1 by Observers",x="Observers", y = "dev1")

# Violin plots of dev2 by Observer
violin_dev2 <- ggplot(devices, aes(x=Observer, y= dev2, fill = Observer)) +
  geom_violin(draw_quantiles = c(0.25, 0.5, 0.75)) +
  labs(title="Violin plots of dev2 by Observers",x="Observers", y = "dev2")

violin_dev1; violin_dev2

# Scatter plots
ggplot(devices, aes(x=dev1,y=dev2)) + geom_point() + facet_wrap(Observer~.)

##### Problem 6 #####
# Need to find  $\int_0^1 e^{-\frac{x^2}{2}} dx$ 
# Let  $f(x) = e^{-\frac{x^2}{2}}$ 
f <- function(x){
  value <- exp(-(x^2/2))
  return(value)
}

# Define a function to find Riemann sum
riemann_sum <- function(f, x, width, method = 'left'){
  # The input f is a function to be integrated
  # The input x is an integration interval
  # The input width is a width of each rectangle
  # The input method is to choose Riemann sum among the left, right or midpoint sum
  if(method == 'left'){
    # Left Riemann sum
    support <- seq(min(x),max(x), by = width) # Support of x
    fx <- f(support[1:(length(support)-1)]) # Height of rectangles
    integral <- width*sum(fx) # Left Riemann sum

  }else if(method == 'right'){
    # Right Riemann sum
    support <- seq(min(x),max(x), by = width) # Support of x
    fx <- f(support[2:length(support)]) # Height of rectangles
    integral <- width*sum(fx) # Right Riemann sum

  }else if(method == 'midpoint'){
    support <- seq(min(x),max(x), by = width) # Support of x
    support <- support - width/2 # To use midpoint as a domain
    fx <- f(support[2:length(support)]) # Height of rectangles
    integral <- width*sum(fx) # Right Riemann sum
  }
}

```



```

}else{
  integral <- c('Not vaild method')
}

return(integral)
}

# Need to find  $\int_0^1 e^{-\frac{x^2}{2}} dx$ 
# Apply different width
rectangle_width = c(.1, .01, .001, .0001)

# Apply various Riemann sum which are left, right and midpoint sums
x <- c(0,1) # Integration interval

result <- c('Left_sum', 'Midpoint_sum', 'Right_sum')
for(i in 1:4){
  result_riemann <-
    c(riemann_sum(f, x, width = rectangle_width[i], method = 'left'),
      riemann_sum(f, x, width = rectangle_width[i], method = 'midpoint'),
      riemann_sum(f, x, width = rectangle_width[i], method = 'right')
    )
  result <- data.frame(result, result_riemann)
}
names(result) <- c('Method', 'Width_0.1', 'Width_0.01', 'Width_0.001', 'Width_0.0001')
result

##### Problem 7 #####

#  $f(x) = 3^x - \sin(x) + \cos(5x)$ 
f <- function(x){
  value <- 3^x - sin(x) + cos(5*x)
  return(value)
}

#  $f'(x) = 3^x \log(3) - \cos(x) - 5\sin(5x)$ 
f_prime <- function(x){
  value <- 3^x*log(3) - cos(x) - 5*sin(5*x)
  return(value)
}

# Define the function to run Newton's method
find_sol_newton <- function(x, tolerance){
  # Input x is the initial value to begin the algorithm, t is tolerance
  # Initial values
  x_new <- x      # Initial value to operate Newton's method
  x_old <- 10000  # Initial value to operate while loop
  no_iter <- 1    # Number of iteration of the loop
  matrix_x <- x   # The matrix of iterated x values to trace the history

  # Newton's Method

```

```

# When x is a vector, break the loop when FALSE for all values in x
while(any(abs(x_new-x_old) > tolerance)){
  x_old <- x_new                                # Update new x value
  x_new <- x_old - f(x_old)/f_prime(x_old)      # Newton's method formula
  matrix_x <- cbind(matrix_x, x_new)           # Store all history of x values
  no_iter <- no_iter + 1                       # Count the number of iteration
}

history_x <- data.frame(t(matrix_x), 1:no_iter) # Stored x history

return(list(Solution = x_new, History_of_x = history_x)) # Roots and histories
}

# Multiple initial values and tolerance
x0 <- -20:20
tolerance <- .0001

# Find roots and history of them using the defined function above
newton_sol <- find_sol_newton(x0,tolerance)

# History of finding roots
# Trace plot of finding solution
# Trace plot of x with the smallest initial value
history_x <- newton_sol$History_of_x            # Recorded history
traceplot <- ggplot(data = history_x) +
  geom_line(aes(x = history_x[,42], y= history_x[,1]), col = 'cornflowerblue') +
  ylim(-30,25) +
  labs(title = 'History of finding roots',
       x = 'Number of iteration', y = 'History')

# Add trace plots of x with the other initial values
for (i in 1:41){
  traceplot <- traceplot + geom_line(aes_string(x = history_x[,42], y= history_x[,i]), col = i)
}

# History of finding roots
traceplot

# Round roots at 4th decimal place and delete replicated solutions
print("The roots"); newton_sol$Solution %>% round(digits = 4) %>% unique()

##### Problem 8 #####
# Define functions to calculate SST
# Calculate SST using a for loop
find_sst_loop <- function(y){
  # The input y is a vector
  sst_element <- numeric(length(y))           # A vector of 0
  for (i in 1:length(y)){
    sst_element[i] <- (y[i] - mean(y))^2      # (Y_i - Y-bar)^2
  }
}

```

```

}
sst <- sum(sst_element)          # SST = sum of (Y_i - Y-bar)^2
return(sst)
}

# Calculate SST using matrix operations only
find_sst_matrix <- function(y){
  # The input y is a vector
  J <- matrix(1, nrow = length(y), ncol = length(y)) # A matrix of 1, J
  sst <- t(y)%*(diag(length(y)) - J/length(y))%*y # SST with matrix notation
  return(sst)
}

# Simulation data
X <- cbind(rep(1,100),rep.int(1:10,time=10))
beta <- c(4,5)
y <- X%*%beta + rnorm(100)

# SST of the functions
find_sst_loop(y)
find_sst_matrix(y)

# Compare operation time using microbenchmark
times <- microbenchmark(result1 <- find_sst_loop(y),
                        result2 <- find_sst_matrix(y),
                        times = 100)
#identical(result1, result2)
autoplot(times)

```