Homework 4

Due Wednesday 9am September 25, 2019

Eric Bae

2019-09-25

For each assignment, turn in by the due date/time. Late assignments must be arranged prior to submission. In every case, assignments are to be typed neatly using proper English in Markdown.

This week, we spoke about R and version control, munging and 'tidying' data, good programming practice and finally some basic programming building blocs. To begin the homework, we will for the rest of the course, start by loading data and then creating tidy data sets.

Problem 1

Work through the "R Programming E" lesson parts 8 and 9.

From the R command prompt:

```
library(swirl)
swirl()
```

Problem 2

Create a new R Markdown file (file–>new–>R Markdown–>save as.

The filename should be: HW4_lastname_firstname, i.e. for me it would be HW4_Settlage_Bob

You will use this new R Markdown file to solve problems 4-10.

Problem 4

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?

In writing functions, the first link, Google's guide to R, recommended that functions be written in Big Camel Case, have a period at the start, not use period in the middle of the function name, and not use "attach()" functions. It was a good read that made me reflect on my coding style because I do virtually all of what it said not to do and not do most of what it tells us to do.
The second link also provided similar guides but with additional details, such as avoiding periods and instead use underline, avoid right-hand assigning, and put more spacing in between formulas and equations. With exception of avoiding periods, I DO follow much of guides here, which makes me proud of myself. So I guess my job is to continue doing what I have been doing correctly.

Problem 5

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*. In this problem, please using *lintr* to lint your last homework (if you didn't do it, perhaps the time is now ;)). In my case, the command looks like this (takes a few moments to run):

```
library(lintr)
lint(filename = "/Users/ericb/Desktop/STAT_5014/HW4_Bae_Eric.Rmd")
```

Can you clean up your code to pass the major issues?? <— just a challenge, not part of the problem!!

Note that really all we have done is syntax checking and received a few stylistic suggestions. We COULD create a custom linter to check for indenting, etc. For now, I think it is enough to know there are a lot of opinions on what code should look like and working in teams requires you to code nicely!! So, clean up your code!!

From the messages, what are some things you need to change in your code?

It opened a "Markers" window but only thing I got was: Line 23 Trailing whitespace is superfluous. Line 24 Trailing whitespace is superfluous. I am going to assume that is a good sign that I am on the right track.

Problem 6

A situation you may encounter is a data set where you need to create a summary statistic for each observation type. Sometimes, this type of redundancy is perfect for a function. Here, we need to create a single function to:

1. calculate the mean for dev1
2. calculate the mean for dev2
3. calculate the sd for dev1
4. calculate the sd for dev2
5. calculate the correlation between dev1 and dev2
6. return the above as a single data.frame

We will use this function to summarize a dataset which has multiple repeated measurements from two devices by thirteen Observers. In the current lecture directory, you will see a file "HW4_data.rds". Please load the file (?readRDS – really nice format for storing data objects), loop through the Observers collecting the summary statistics via your function for each Observer separately.

The output of this problem should be:

a. A single table of the means, sd, and correlation for each of the 13 Observers

b. A box plot of all the means to compare the spread of means from dev1 to dev2

c. A violin plot of all the sd to compare the spread of sd from dev1 to dev2

I will look at the code and comment on it, so make it NICE!!

```
Dat <- readRDS("/Users/ericb/Desktop/STAT_5014/Hw4_data.rds")

Summary_Stat <- function(dat) {
  # Download packages
  if (!("tidyr" %in% installed.packages())) {
    install.packages("tidyr")
  }
  if (!("dplyr" %in% installed.packages())) {
    install.packages("dplyr")
  }
  if (!("ggplot2" %in% installed.packages())) {
    install.packages("dplyr")
  }
  if (!("gridExtra" %in% installed.packages())) {
```

```
  install.packages("dplyr")
}

# Load Packages
require(tidyr)
require(dplyr)
require(ggplot2)
require(gridExtra)

# Tibbles
dat_tibble <- as_tibble(dat)
dat_summary <- dat_tibble %>%
  group_by(Observer) %>%
  summarize(`Mean of dev1` = mean(dev1),
            `Mean of dev2` = mean(dev2),
            `SD of dev1` = sd(dev1),
            `SD of dev2` = sd(dev2),
            Correlation = cor(dev1, dev2))

# Convert to data frame
dat_summary_df <- as.data.frame(dat_summary)

# Organize data for plotting
dev1_type <- rep("dev1", nrow(dat_summary_df))
dev2_type <- rep("dev2", nrow(dat_summary_df))

# Subsetting means from original data frame
dev1_mean_df <- cbind.data.frame(dat_summary_df$`Mean of dev1`, dev1_type)
dev2_mean_df <- cbind.data.frame(dat_summary_df$`Mean of dev2`, dev2_type)
colnames(dev1_mean_df) <- colnames(dev2_mean_df) <- c("Mean", "Dev type")

# Subsetting SD from original data frame
dev1_sd_df <- cbind.data.frame(dat_summary_df$`SD of dev1`, dev1_type)
dev2_sd_df <- cbind.data.frame(dat_summary_df$`SD of dev2`, dev2_type)
colnames(dev1_sd_df) <- colnames(dev2_sd_df) <- c("SD", "Dev type")

# Box plots of means
dev1_box <- ggplot(dev1_mean_df, aes(x = `Dev type`, y = `Mean`)) +
  geom_boxplot(fill = "hotpink") +
  ggtitle("Boxplot of mean of dev1")
dev2_box <- ggplot(dev2_mean_df, aes(x = `Dev type`, y = `Mean`)) +
  geom_boxplot(fill = "skyblue") +
  ggtitle("Boxplot of mean of dev2")

# Violin plots of SDs
dev1_violin <- ggplot(dev1_sd_df,
                      aes(x = `Dev type`, y = `SD`)) +
  geom_violin(fill = "yellowgreen") +
  ggtitle("Violin plot of SD of dev1")
dev2_violin <- ggplot(dev2_sd_df,
                      aes(x = `Dev type`, y = `SD`)) +
  geom_violin(fill = "gold") +
  ggtitle("Violin plot of SD of dev1")
```

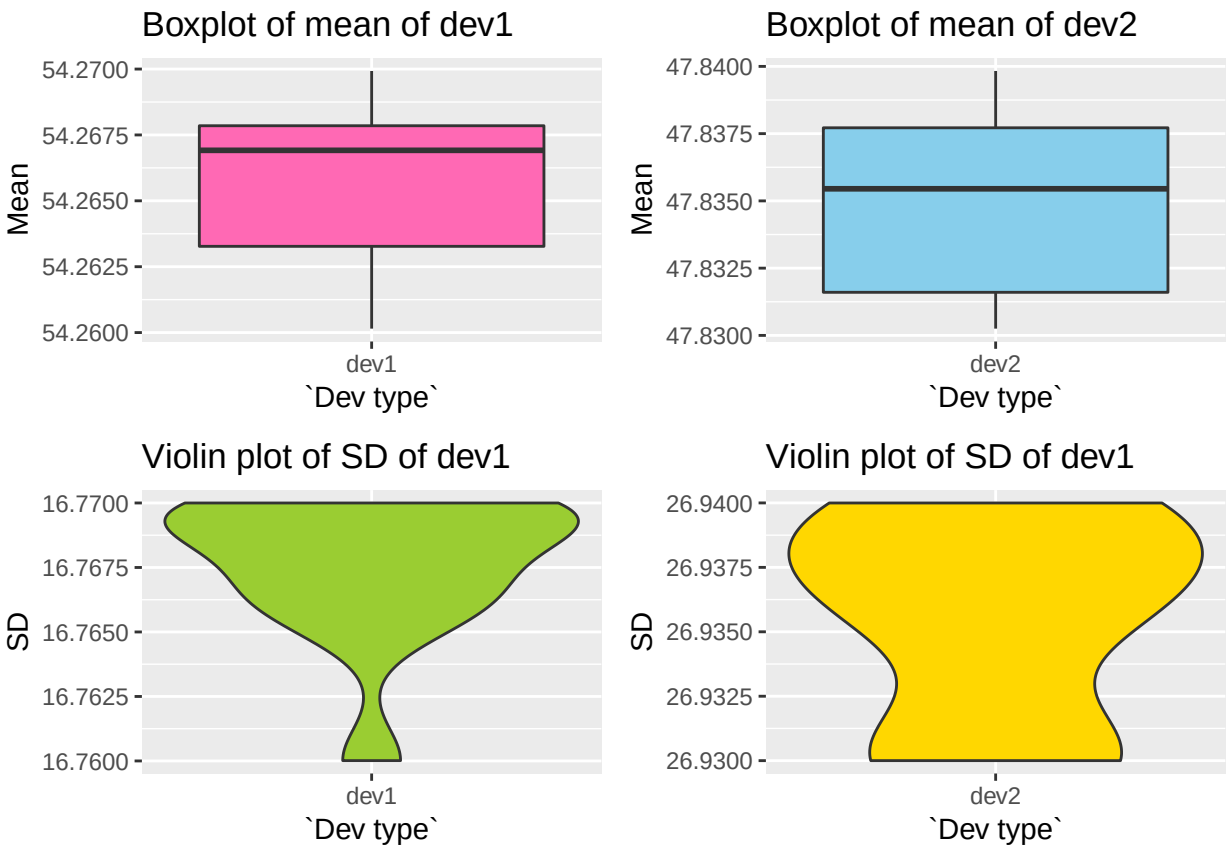
```
  grid.arrange(dev1_box, dev2_box,
               dev1_violin, dev2_violin, nrow = 2)

  return(dat_summary_df)
}

# Run our function on the data set
Summary_Stat(Dat)
```

```
## Warning: `as_dictionary()` is soft-deprecated as of rlang 0.3.0.
## Please use `as_data_pronoun()` instead
## This warning is displayed once per session.
```

```
## Warning: `new_overscope()` is soft-deprecated as of rlang 0.2.0.
## Please use `new_data_mask()` instead
## This warning is displayed once per session.
```

```
## Warning: The `parent` argument of `new_data_mask()` is deprecated.
## The parent of the data mask is determined from either:
##
##   * The `env` argument of `eval_tidy()`
##   * Quosure environments when applicable
## This warning is displayed once per session.
```

```
## Warning: `overscope_clean()` is soft-deprecated as of rlang 0.2.0.
## This warning is displayed once per session.
```



```
##    Observer Mean of dev1 Mean of dev2 SD of dev1 SD of dev2 Correlation
## 1         1     54.26610     47.83472   16.76982   26.93974 -0.06412835
## 2         2     54.26873     47.83082   16.76924   26.93573 -0.06858639
## 3         3     54.26732     47.83772   16.76001   26.93004 -0.06834336
## 4         4     54.26327     47.83225   16.76514   26.93540 -0.06447185
## 5         5     54.26030     47.83983   16.76774   26.93019 -0.06034144
## 6         6     54.26144     47.83025   16.76590   26.93988 -0.06171484
## 7         7     54.26881     47.83545   16.76670   26.94000 -0.06850422
## 8         8     54.26785     47.83590   16.76676   26.93610 -0.06897974
## 9         9     54.26588     47.83150   16.76885   26.93861 -0.06860921
## 10       10     54.26734     47.83955   16.76896   26.93027 -0.06296110
## 11       11     54.26993     47.83699   16.76996   26.93768 -0.06944557
## 12       12     54.26692     47.83160   16.77000   26.93790 -0.06657523
## 13       13     54.26015     47.83972   16.76996   26.93000 -0.06558334
```

Problem 7

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

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

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.

Note: use good programming practices.

```r
riemann_sum <- function(width) {
  # Define attributes
  est_sum <- sum_error <- rep(NA, length(width))
  sum_error2 <- 2
  ub <- 1 # Upper bound
  lb <- 0 # Lower bound

  # True density
  true_solution <- sqrt(2 * pi) * (pnorm(ub) - pnorm(lb))
  k <- 1

  # Loop to generate estimated sum and error
  while (abs(sum_error2) > 1e-6) {
    width.k <- width[k]
    lw_lim <- seq(lb, ub, width.k)
    lw_lim <- lw_lim[-length(lw_lim)]
    up_lim <- seq(lb, ub, width.k)[-1]
    mid_point <- (lw_lim + up_lim) / 2

    F_x <- exp(-mid_point ^ 2 / 2)
    est_sum[k] <- sum(width.k * F_x)
    sum_error[k] <- true_solution - est_sum[k]
    sum_error2 <- sum_error[k]
```

```
    necessary_width <- width.k
    k <- k + 1
  }

  # Creating summary table
  summary_table <- cbind.data.frame(width[1:k], est_sum[1:k],
                                    sum_error[1:k])
  rownames(summary_table) <- 1:k
  colnames(summary_table) <- c("Widths Tested", "Estimated Sum",
                               "Riemann Sum Error")
  summary_table <- round(summary_table, 7)

  return(list(summary_table = summary_table,
              width = width, est_sum = est_sum,
              necessary_width = necessary_width))
}

# Testing my fuction with various widths
width_tested <- c(1, cumprod(rep(1 / 2, 50)))
riemann_summary <- riemann_sum(width_tested)
riemann_summary$summary_table
```

```
##    Widths Tested Estimated Sum Riemann Sum Error
## 1      1.0000000     0.8824969        -0.0268725
## 2      0.5000000     0.8620364        -0.0064120
## 3      0.2500000     0.8572097        -0.0015853
## 4      0.1250000     0.8560196        -0.0003952
## 5      0.0625000     0.8557231        -0.0000987
## 6      0.0312500     0.8556491        -0.0000247
## 7      0.0156250     0.8556306        -0.0000062
## 8      0.0078125     0.8556259        -0.0000015
## 9      0.0039062     0.8556248        -0.0000004
## 10     0.0019531            NA                NA
```

```
riemann_summary$necessary_width
```

```
## [1] 0.00390625
```

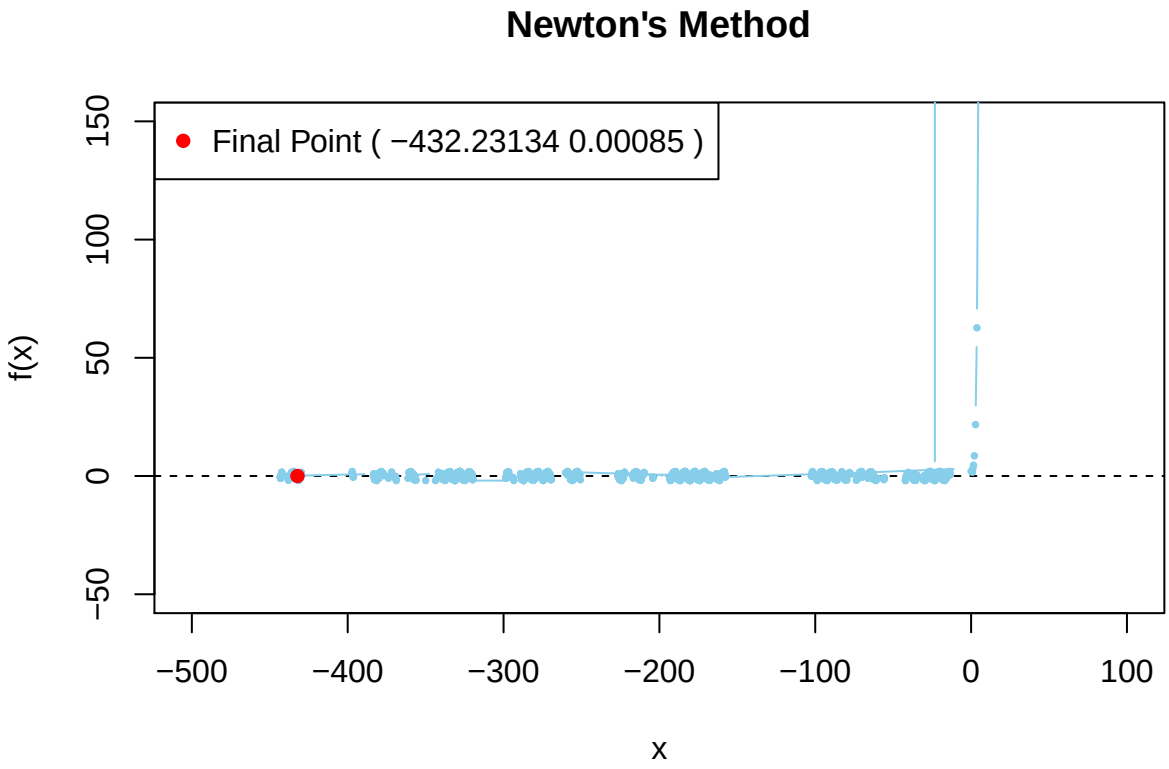
Problem 8

Create a function to find solutions to (1) using Newton's method. The answer should include the solutions with tolerance used to terminate the loop, the interval used, and a plot showing the iterations on the path to the solution.

$$f(x) = 3^x - sin(x) + cos(5x) \tag{1}$$

```
newtons_method <- function(init_value, tolerance = 1e-3) {
  x0 <- init_value
  f_x0 <- 3 ^ x0 - sin(x0) + cos(5 * x0)
  df_x0 <- 3 ^ x0 * log(3) - cos(x0) + 5 * sin(5 * x0)
  plot(x0, f_x0, pch = 16, cex = 0.5, col = "skyblue",
```

```r
       xlim = c(-500, 100), ylim = c(-50, 150),
       xlab = "x", ylab = "f(x)", main = "Newton's Method")
  abline(h = 0, lty = 2)

  # Initial values set up
  x_points <- matrix(0, nrow = 2, ncol = 2)
  x_interval <- c(x0, x0)
  y_interval <- c(f_x0, f_x0)

  trial <- 0

  while ( (abs(f_x0) > tolerance) | (trial < 1000) ) {
    x_points[1, ] <- c(x0, f_x0)
    x1 <- x0 - f_x0 / df_x0
    x0 <- x1
    f_x0 <- 3 ^ x0 - sin(x0) + cos(5 * x0)
    df_x0 <- 3 ^ x0 * log(3) - cos(x0) + 5 * sin(5 * x0)

    # Updating information
    x_points[2, ] <- c(x0, f_x0)
    x_interval <- c(min(x0, min(x_interval)), max(x0, max(x_interval)))
    y_interval <- c(min(f_x0, min(y_interval)), max(f_x0, max(y_interval)))
    points(x_points, pch = 16, cex = 0.5, col = "skyblue", type = "b")

    trial <- trial + 1
  }

  points(x_points[2, 1], x_points[2, 2], pch = 16, col = "red")
  x_points <- round(x_points, 5)
  legend("topleft",
         legend = paste("Final Point", "(",
                        x_points[2, 1], x_points[2, 2], ")"),
         pch = 16, col = "red")

  return(list(root = x0, x_interval = x_interval,
              y_interval = y_interval, tolerance = tolerance))
}

# Testing my value
newtons_method(0)
```




```
## $root
## [1] -432.2313
##
## $x_interval
## [1] -443.30678   38.35821
##
## $y_interval
## [1] -1.952894e+00  2.002253e+18
##
## $tolerance
## [1] 0.001
```

Problem 9

Finish this homework by pushing your changes to your repo.

Only submit the .Rmd and .pdf solution files. Names should be formatted HW4_lastname_firstname.Rmd and HW4_lastname_firstname.pdf

Optional preperation for next class:

Next week we will talk about Exploratory Data Analysis and graphing. Swirl will be a bit part of this. Check out "Exploratory_Data_Analysis" Swirl lessons 1-10.