HW3_ahcooper

Andrew Cooper

9/28/2020

Problem 3

The guidelines for good programming style outlined in the style guides are designed to minize the ambiguity of R code and maintain a level of consistency in style. The R language can often be forgiving in what kinds of code it accepts as "errorless", which can allow users to code in ambiguous and "improper" ways. I think these style guides help enforce a stricter styler in function-creation and implementation that should make reviewing or revising code significantly easier.

As for my programming practices, I plan to be more diligent in the style of variable naming, as I have been inconsistent with variable naming in the past.

Problem 5

```
DataframeSummarize <- function(df){
  # Takes in two-column dataframe as input, outputs vector of summary statistics
  return(c(
    sapply(df, mean),
    sapply(df, sd),
    cor(df[,1], df[,2]),
    use.names = F
  ))
}
```

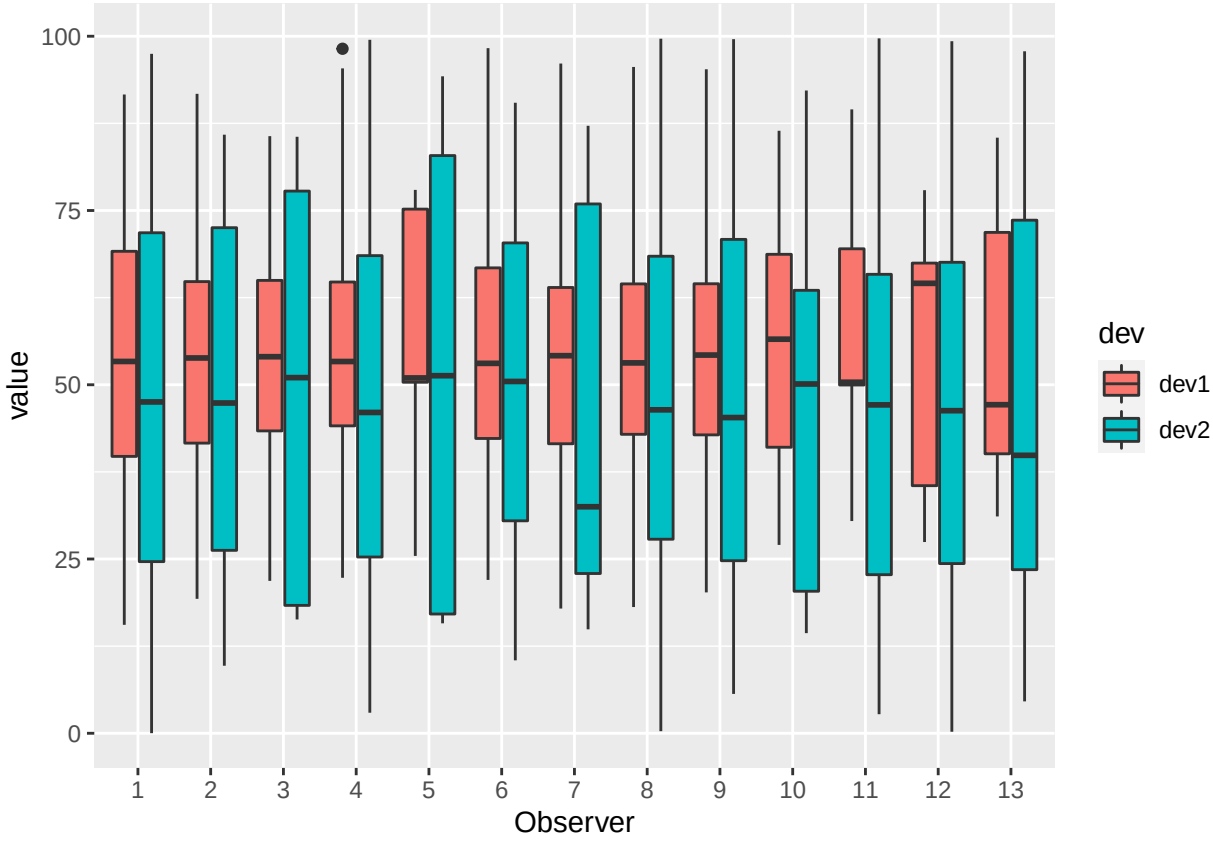
```
# Read in dataframe
df <- readRDS("HW3_data.rds")
```

```
sum_table <- matrix(nrow = 13, ncol = 5)
unique_obs <- sort(unique(df$Observer))

# Iterate through each observer
for(obs in unique_obs){
  obs_df <- df[which(df$Observer == obs), -1]
  sum_table[obs, ] <- DataframeSummarize(obs_df)
}

# Store summaries in data frame
sum_table <- cbind(unique_obs, sum_table)
sum_table <- as.data.frame(sum_table)
```

```
# Output table
colnames(sum_table) <- c("Observer", "dev1_mean", "dev2_mean", "dev1_sd", "dev2_sd", "corr(dev1, dev2)")
kable(sum_table)
```

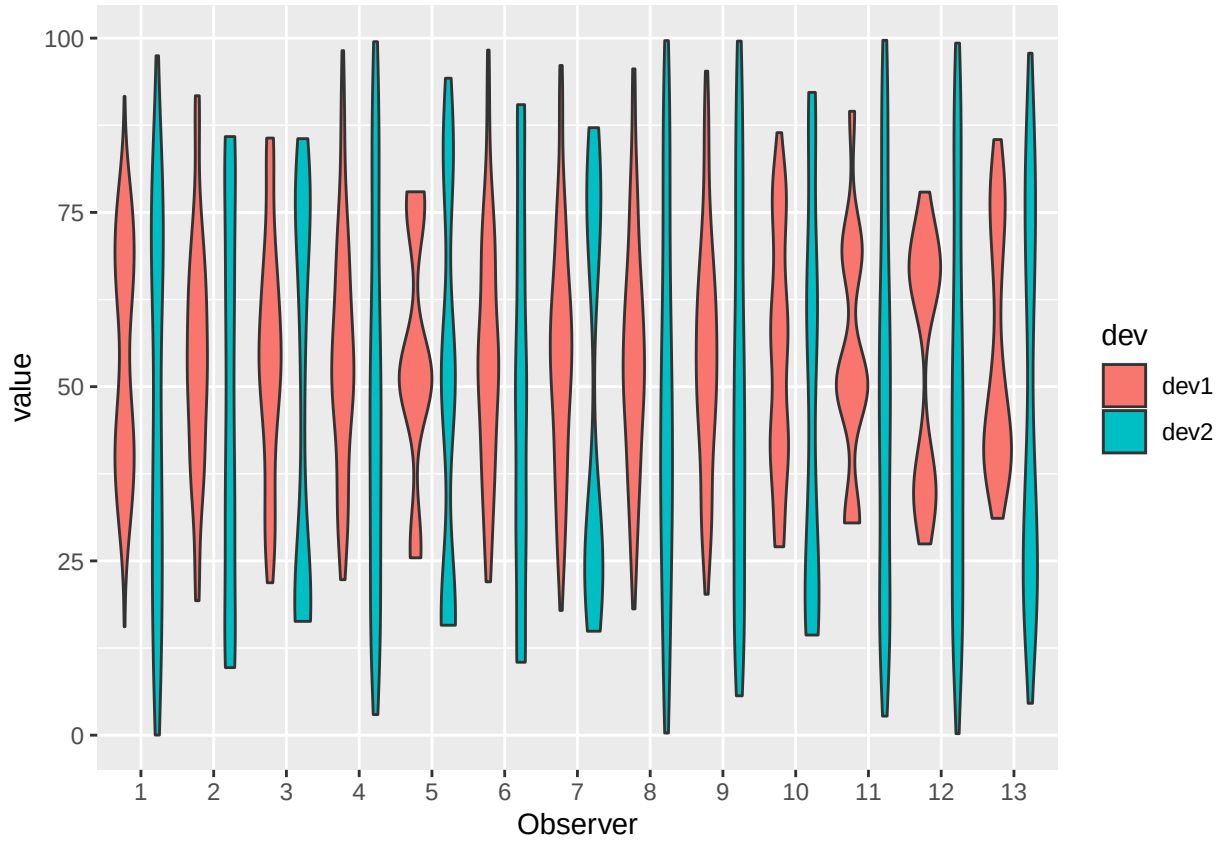
Observer	dev1_mean	dev2_mean	dev1_sd	dev2_sd	corr(dev1, dev2)
1	54.26610	47.83472	16.76983	26.93974	-0.0641284
2	54.26873	47.83082	16.76924	26.93573	-0.0685864
3	54.26732	47.83772	16.76001	26.93004	-0.0683434
4	54.26327	47.83225	16.76514	26.93540	-0.0644719
5	54.26030	47.83983	16.76774	26.93019	-0.0603414
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

```
# Side-by-side boxplots of dev1 and dev2
df %>%
  mutate(Observer = factor(Observer)) %>%
  gather(dev, value, -Observer) %>%
  ggplot(aes(x = Observer, y = value, fill = dev)) +
  geom_boxplot()
```



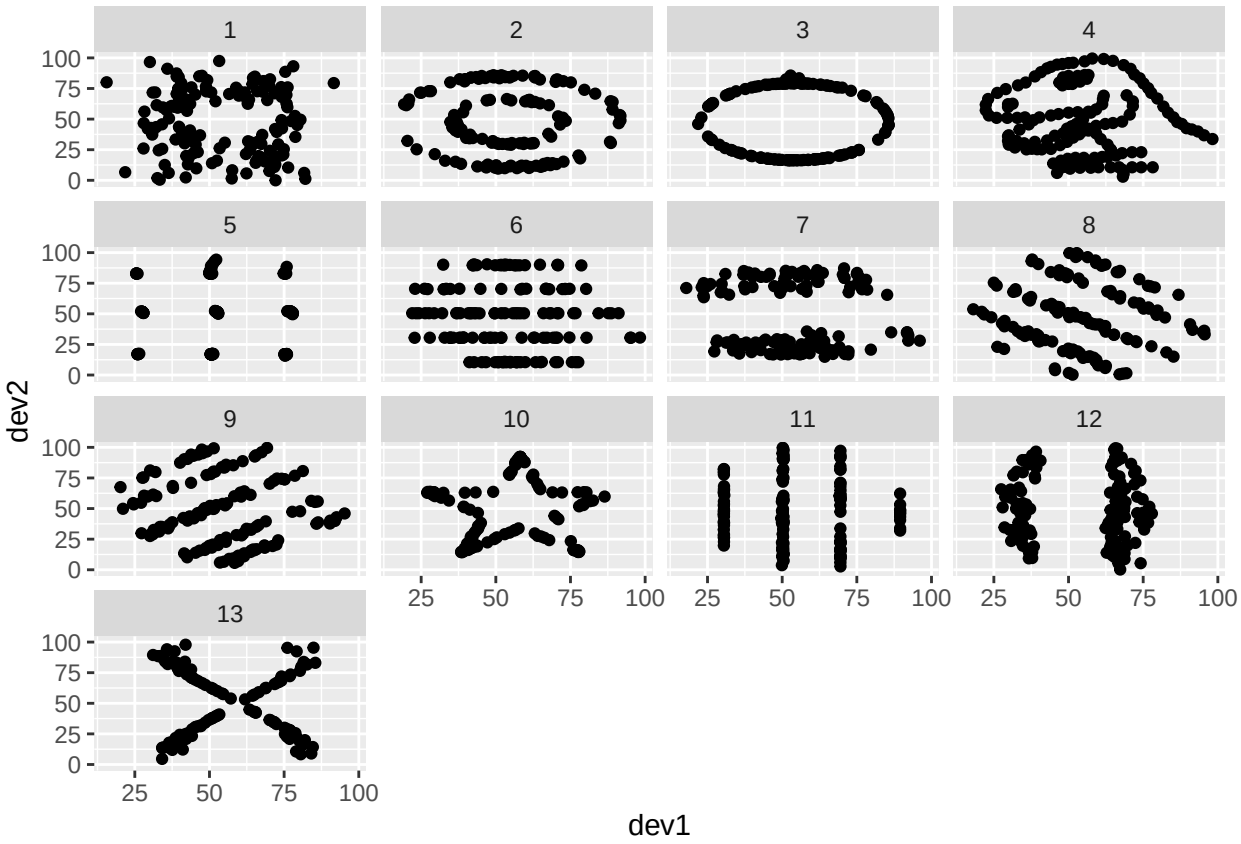

```
# Side-by-side violin plots of dev1 and dev2
df %>%
  mutate(Observer = factor(Observer)) %>%
  gather(dev, value, -Observer) %>%
```

```
  ggplot(aes(x = Observer, y = value, fill = dev)) +
  geom_violin()
```



The plots seem to hint at some significant differences in dev1 and dev2 among observers.

```
# Scatterplots of  dev1 and dev2
df %>%
  ggplot(aes(x = dev1, y = dev2)) +
  geom_point() +
  facet_wrap(Observer~.)
```



The scatterplots for each observer shows that for all of the observers there is a clear non-linear relationship between dev1 and dev2, where some even have the shape of drawings like a dinosaur or a star. This shows how basic visual data exploration like scatterplots can reveal key features about the data that summary statistics simply cannot. Had we not created these scatterplots we could have easily concluded linear relationships among the data that clearly do not exist.

Problem 6

```
Riemman <- function(w){
  # Computes a left-hand Riemann sum to estimate the integral of exp(-x^2/2) from 0 to 1, with rectangl
  x <- seq(0, 1, w)
  f <- function(x){exp(-x^2/2)}
  return(sum(f(x)*w))
}
```

```
true_val <- 0.855624
tol <- 1e-6
w <- 1
diff <- 1
# Divide width by 10 until Riemman sum error is below 1*10^-6
while(diff > tol){
  w <- w / 10
  diff <- abs(Riemman(w) - true_val)
}

paste0("Width = ", w, ", Estimate = ", round(Riemman(w), 3), ", Error = ", round(diff, 9))
```

```
## [1] "Width = 1e-07, Estimate = 0.856, Error = 4.12e-07"
```

I found the width necessary to estimate the integral with an error less than 1*10^-6 by starting with a width of 1, then dividing that width by 10 until the error was small enough.

Problem 7

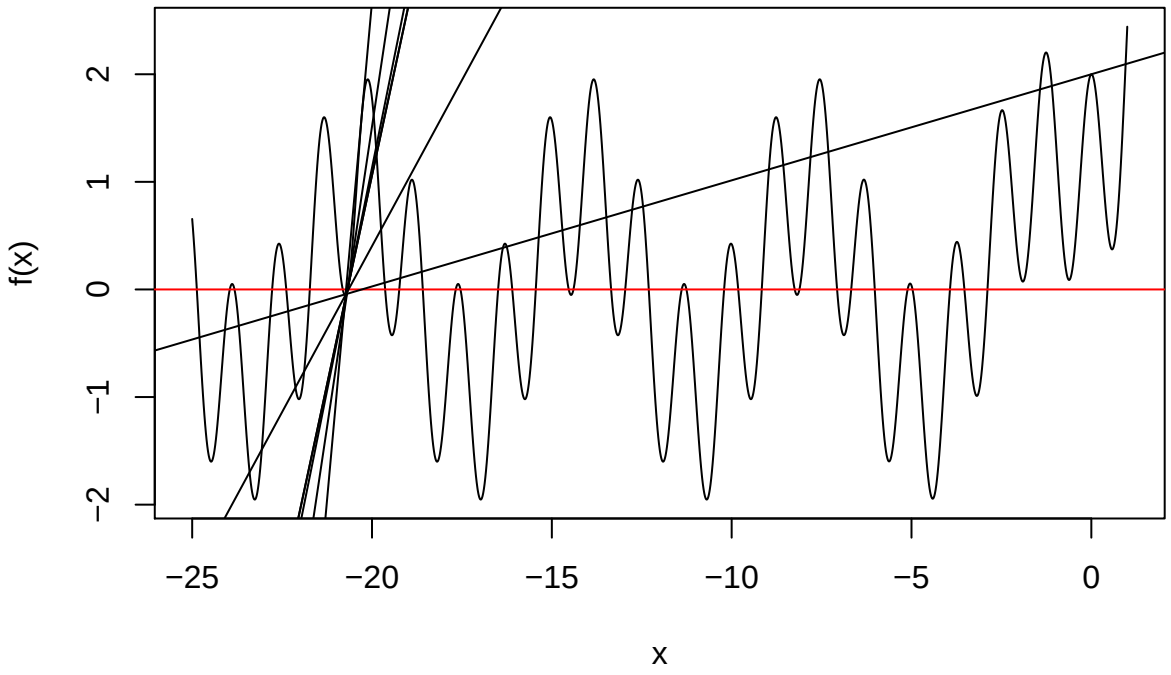
```
Newton <- function(f, fp, x0, tol = 1e-6){
  # Returns a root of the function f with derivative fp at starting point x0 under error tolerance tol
  x <- x0
  x_vals <- c(x)
  f_vals <- c(f(x))
  fp_vals <- c(fp(x))
  diff <- abs(f(x))
  diff_vals <- c(diff)
  # Iterate until y-value at x is below tolerance
  while(diff > tol){
    x <- x - f(x)/fp(x)
    diff <- abs(f(x))
    diff_vals <- c(diff_vals, diff)
    x_vals <- c(x_vals, x)
    f_vals <- c(f_vals, f(x))
    fp_vals <- c(fp_vals, fp(x))
  }
  return(list(
    "root" = x,
    "iterations" = diff_vals,
    "x_vals"     = x_vals,
    "f_vals"     = f_vals,
    "fp_vals"    = fp_vals
  ))
}
```

```
f <- function(x){3^x - sin(x) + cos(5*x)}
fp <- function(x){3^x*log(3) - cos(x) - 5*sin(5*x)}
newton_results <- Newton(f, fp, 0)
paste0("Root = ", newton_results$root, "; Number of Iterations = ", length(newton_results$iterations))
```

```
## [1] "Root = -20.6821514805075; Number of Iterations = 7"
```

Our function found a root for the function at around x = -20.68215

```
x <- seq(-25, 1, 0.01)
plot(x, f(x), type = "l")
lines(abline(a = 0, b = 0, col = "red"))
x_vals <- newton_results$x_vals
f_vals <- newton_results$f_vals
fp_vals <- newton_results$fp_vals

for(i in 1:length(x_vals)){
  lines(abline(a = (f_vals[i] - fp_vals[i]*x_vals[i]), b = fp_vals[i]))
}
```



Problem 8

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

```
SSE <- 0
for(e in y){
  SSE <- SSE + (e - ybar)^2
}
SSE
```

```
## [1] 20809.41
```

```
microbenchmark({
  SSE <- 0
  for(e in y){
    SSE <- SSE + (e - ybar)^2
  }})
```

```
## Unit: milliseconds
##                                                                                 expr
##  {     SSE <- 0     for (e in y) {         SSE <- SSE + (e - ybar)^2     } }
##       min       lq     mean   median       uq      max neval
##  2.385257 2.446219 3.243364 2.528189 2.780455 33.69588   100
```

```
r <- y - ybar
SSE <- SSE <- t(r) %*% r
SSE
```

```
##          [,1]
## [1,] 20809.41
```

```r
microbenchmark({
  r <- y - ybar
  SSE <- SSE <- t(r) %*% r
}) %>% print()
```

```
## Unit: microseconds
##                                                expr   min     lq    mean median
##  {     r <- y - ybar     SSE <- SSE <- t(r) %*% r } 3.237 3.3295 3.71283  3.414
##       uq    max neval
##  3.5825 26.775   100
```