HW3_xinyis

Xinyi Song

9/24/2020

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.

The last couple of weeks, we spoke about R, version control and Reproducible Research, munging and 'tidying' data, good programming practice, some basic programming building blocs, and finally matrix/vector operations. In this homework, we will put this all together and actually analyze some data. 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

In the "Getting and Cleaning Data" lesson set, you should be comfortable with lessons 1-3. Work through the "R Programming E" lesson as you see fit. Lessons 1-9 and 15 are ones you should consider paying special attention to. If you prefer the Rstudio.cloud Primers, the Primer on "Write Functions" is well done.

From the R command prompt:

```r
library(swirl)
install_course("R_Programming_E")
install_course("Getting_and_Cleaning_Data")
install_course("Exploratory_Data_Analysis")
swirl()
```

Problem 2

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

The filename should be: HW3_pid, i.e. for me it would be HW3_rsettlag

You will use this new R Markdown file to solve the following problems:

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?

Try to use sapply/apply functions instead of loop for higher efficiency Make Code Self-Documenting Give code comments

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

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 which takes as input a two column dataframe 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

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

We will use this function to summarize a dataset which has multiple repeated measurements from two devices (dev1 and dev2) by thirteen Observers. This file is preformatted as an R object, so it will read in nicely. "url <- https://github.com/rsettlage/STAT_5014_Fall_2020/blob/master/homework/HW3_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 and store in a single dataframe.

The output of this problem should be:

a. A single table of the means, sd, and correlation for each of the 13 Observers (*?kable*). From this table, what would you conclude? You can easily check your result using dplyr's group_by and summarize.

Solution

```r
setwd("~/Desktop/VT Course/STAT 5014")
dat <- readRDS("HW3_data.rds")
My_function = function(data){
obs = unique(data[,1])
summary_tab = matrix(0, length(obs), 6)
for (i in 1:length(obs)){
t = data[which(data[,1]==obs[i]), ]
summary_tab[i,1] = obs[i]
summary_tab[i,2] = mean(t[,2])
summary_tab[i,3] = mean(t[,3])
summary_tab[i,4] = sd(t[,2])
summary_tab[i,5] = sd(t[,3])
summary_tab[i,6] = cor(t[,2], t[,3])
}
summary_tab = summary_tab[order(summary_tab[,1]),]
colnames(summary_tab) = c('Observer', 'mean_dev1', 'mean_dev2', 'sd_dev1', 'sd_dev2', 'correlation')
return(summary_tab)
}
# Call my function
summary_tab_myfunc = My_function(dat)
knitr::kable(summary_tab_myfunc)
```

Observer	mean_dev1	mean_dev2	sd_dev1	sd_dev2	correlation
1	54.26610	47.83472	16.76983	26.93974	-0.0641284

Observer	mean_dev1	mean_dev2	sd_dev1	sd_dev2	correlation
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

```r
# Check my result with dplyr's group_by and summarize
library(dplyr, warn.conflicts = FALSE)
# Suppress summarise info
options(dplyr.summarise.inform = FALSE)
group_table = dat %>%
group_by(Observer)
summary_table = group_table %>% summarize(mean_dev1 = mean(dev1), mean_dev2 = mean(dev2), sd_dev1 = sd(
          correlation = cor(dev1, dev2))
knitr::kable(summary_table)
```

Observer	mean_dev1	mean_dev2	sd_dev1	sd_dev2	correlation
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

Here, 'summary_tab_myfunc' is the output of my function while 'summary_table' is the output from dplyr-group and summarize function. The outputs are same.

Based on the outputs above, I can see that for each observer, the correlation between dev1 and dev2 is around -0.07, which indicates that for observer 1 to 13, dev1 and dev2 in general has weak and negative correlation.

Besides, for value dev1, the mean value of each observer is around 47.83. Observer 11 has the largest mean value of dev1 with 54.26993 while Observer 13 has smallest value of dev1 with 54.26015. For their standard deviation, there are not much differences, all are around 16.77.

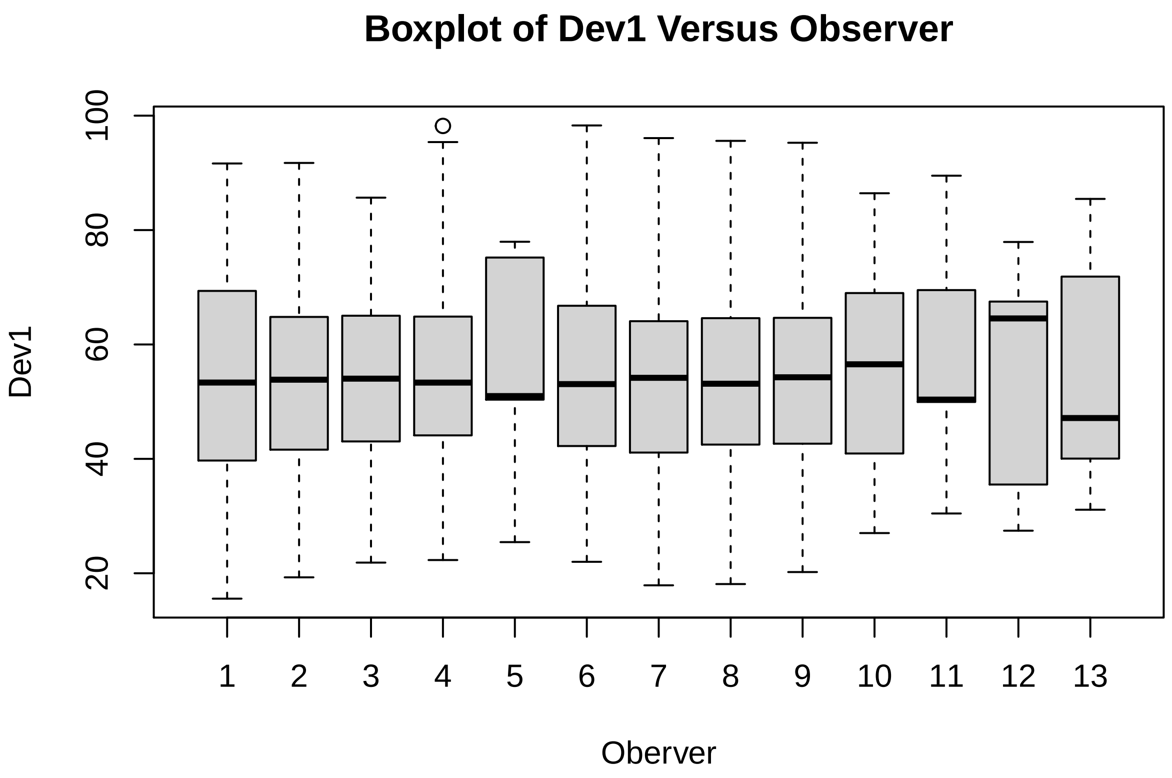
for value dev2, the mean value of each observer is around 54.26. Observer 13 has the largest mean value of dev2 with 47.83972 while Observer 6 has smallest value of dev2 with 47.83025. For their standard deviation, there are not much differences, all are around 26.93.

In general, we can see that although the magnitude of dev1 is larger than that of dev2, the standard deviation of dev1 is much smaller than that of dev2.

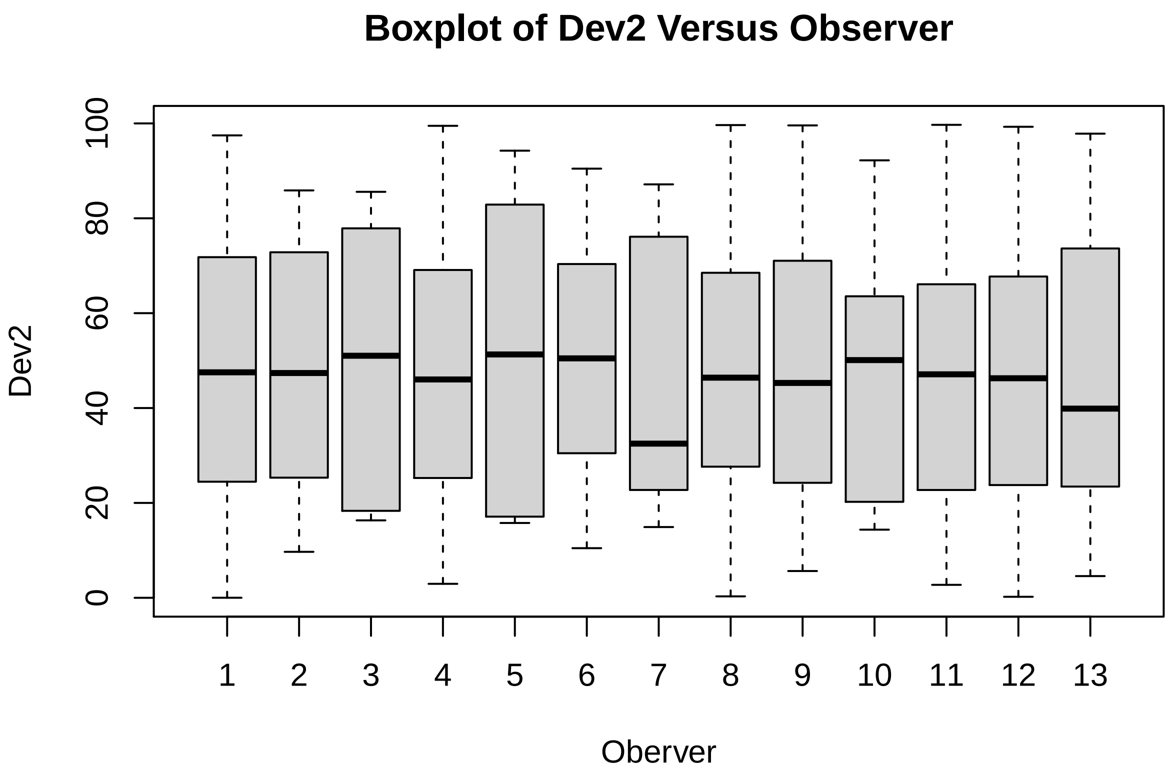
b. A box plot of dev, by Observer (*?boxplot*). From these plots, what would you conclude?

Solution

```
# Box Plot
data = dat[order(dat[,1]),]
box_dev1 = data.frame(values = data[,2], vars = rep(c(1:length(unique(dat$Observer))), each = dim(dat)[
boxplot(values ~ vars, data =box_dev1, xlab = 'Oberver', ylab = 'Dev1', main = 'Boxplot of Dev1 Versus 
```



```
box_dev2 = data.frame(values = data[,3], vars = rep(c(1:length(unique(dat$Observer))), each = dim(dat)[
boxplot(values ~ vars, data =box_dev2, xlab = 'Oberver', ylab = 'Dev2', main = 'Boxplot of Dev2 Versus 
```



Based on the results above, we can see that for value of dev1, the distriution of dev1 of Observer 12 is negative skew (skewed left) since the median is closer to the top of the box and the whisker is shorter on the upper end of the box. On the contrary, the dev1 of Observer 5, 11 and 13 is positive skew (skewed right) since median is closer to the bottom of the box, and the whisker is shorter on the lower end of the box. The distribution of dev1 of other observers look symmetric.

For value of dev2, the distriution of dev2 of Observer 10 is negative skew (skewed left) since the median is closer to the top of the box and the whisker is shorter on the upper end of the box. On the contrary, the dev2 of Observer 7 and 13 is positive skew (skewed right) since median is closer to the bottom of the box, and the whisker is shorter on the lower end of the box. The distribution of dev2 of other observers look symmetric.

c. A violin plot of dev by Observer (*??violin* two "?" will search through installed packages). From these plots, what would you conclude? Compared to the boxplot and summary statistics, thoughts?

Solution

```r
# Violin Plot
library(ggplot2)
library(dplyr)
library(hrbrthemes)
```

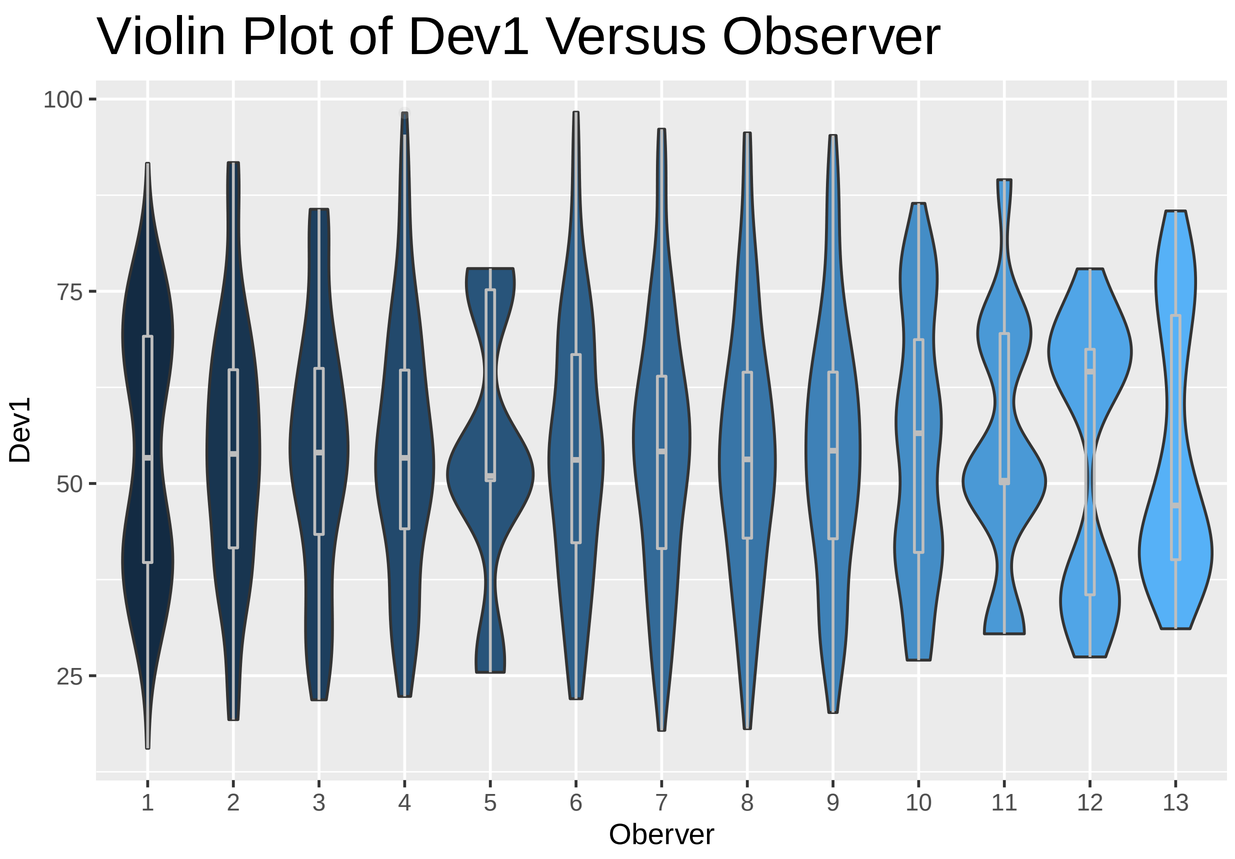
```
## NOTE: Either Arial Narrow or Roboto Condensed fonts are required to use these themes.

##       Please use hrbrthemes::import_roboto_condensed() to install Roboto Condensed and

##       if Arial Narrow is not on your system, please see https://bit.ly/arialnarrow
```

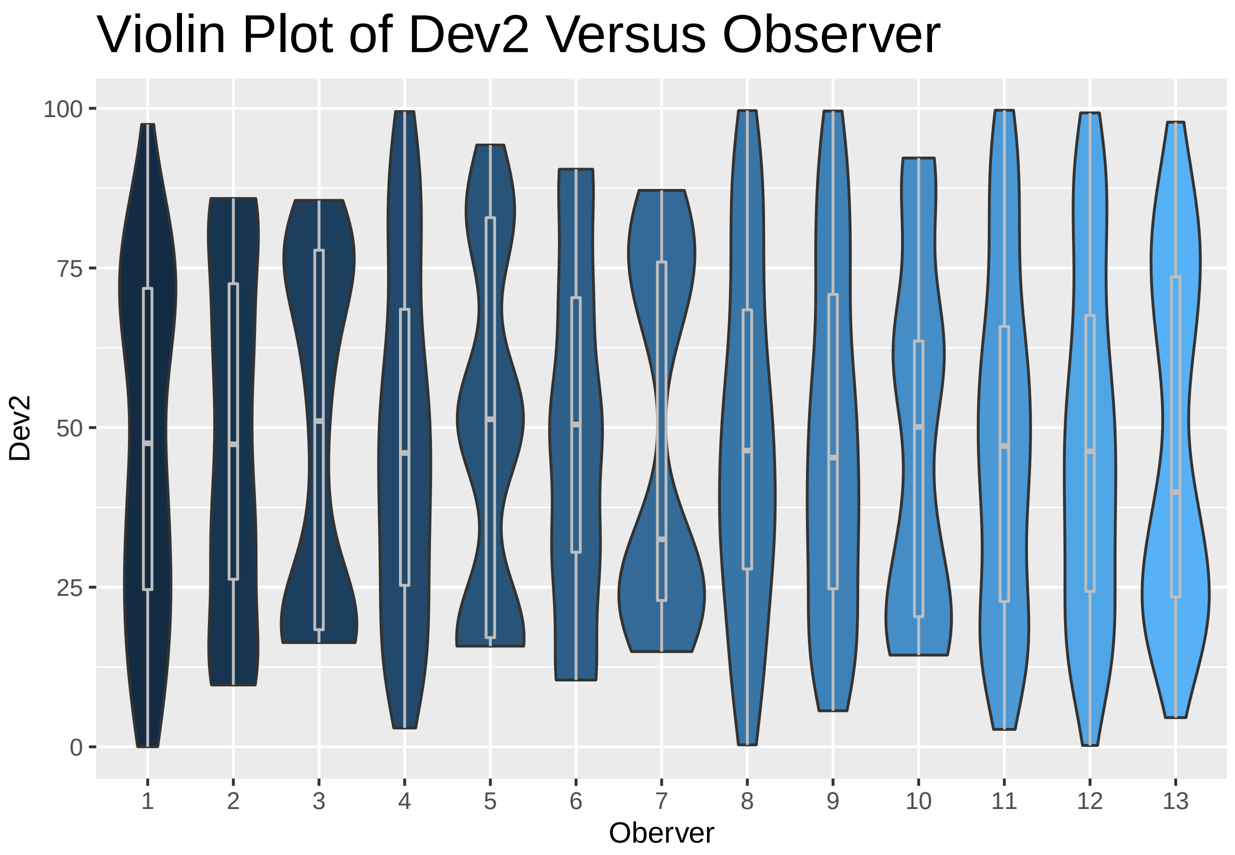
```r
data = dat[order(dat[,1]),]
Observer_size = data %>% group_by(Observer)%>% summarize(num=n())
data %>% left_join(Observer_size) %>%
  mutate(myaxis = paste0(num))%>%
  ggplot(aes(x=reorder(myaxis, x = Observer), y = dev1, fill= Observer)) +        geom_violin(width =
  geom_boxplot(width=0.1, color="grey", alpha=0.2) +
  theme(
    legend.position="none",
    plot.title = element_text(size = 20)
  ) +
  ggtitle("Violin Plot of Dev1 Versus Observer") +
  xlab("Oberver") + ylab("Dev1")
```

```
## Joining, by = "Observer"
```



```
Observer_size = data %>% group_by(Observer)%>% summarize(num=n())
data %>% left_join(Observer_size) %>%
  mutate(myaxis = paste0(num))%>%
  ggplot(aes(x=reorder(myaxis, x = Observer), y = dev2, fill= Observer)) + geom_violin(width = 1)  +
  geom_boxplot(width=0.1, color="grey", alpha=0.2) +
  theme(
    legend.position="none",
    plot.title = element_text(size = 20)
  ) +
  ggtitle("Violin Plot of Dev2 Versus Observer") +
  xlab("Oberver") + ylab("Dev2")
```

```
## Joining, by = "Observer"
```



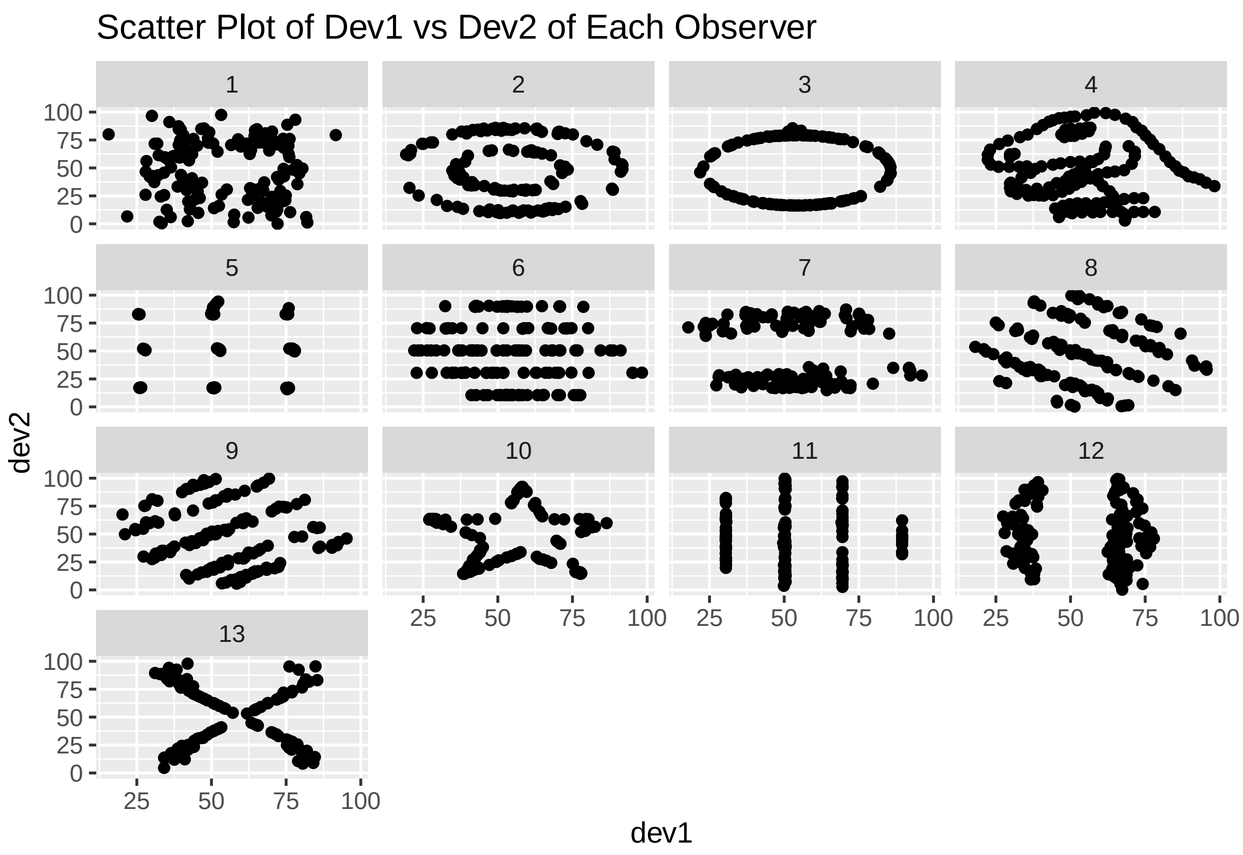
Now that you have made some conclusions and decided what your analysis may look like, you decide to make one more plot:

The violin plot provides similar statistics such as median, interquartile range and the lower/upper adjacent values. However, the 'violin' shape of a violin plot comes from the data's density plot, which displays frequencies of values. We could get the same conclusion about the summary statistics as boxplot, besides, violin plot provides additional information about kernel density information.

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

`Solution`

```r
# Scatter Plot
ggplot(data, aes(x=dev1,y=dev2)) + geom_point() + facet_wrap(Observer~.) +
  ggtitle("Scatter Plot of Dev1 vs Dev2 of Each Observer")
```



What do you see? Combining the scatter plot with the summary statistics, what is the lesson here? As you approach data analysis, what things should you do in the "Exploratory Data Analysis" portion of a project to avoid embarrassment from making erroneos conclusions?

Comments: This scatter plot matrix shows the correlation between dv1 and dv2 for each observer. There is no clear shape for them, which matches the statistics correlation around -0.06, extremely weak negative correlation, we could hardly find it in the plot.

Problem 6

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. For help on Reimann sums: https://www.khanacademy.org/math/ap-calculus-ab/ab-integration-new/ab-6-2/a/left-and-right-riemann-sums

```
f = function(x){
 results = exp(-1/2*x^2)
 return(results)
```

```r
}
Reimann_function = function(f, lower_bound, upper_bound, width){
dx = (upper_bound - lower_bound)/width
grid = seq(0, 1, by = width)
tmp = 0
for (i in 1:length(grid)){
tmp = tmp + f(grid[i])*width
}
return(tmp)
}
# Call my function
# Width = 0.001
my_calculation_1 = Reimann_function(f, 0, 1, 0.001)
# Width = 0.00001
my_calculation_2 = Reimann_function(f, 0, 1, 0.00001)
# Width = 0.0000001
my_calculation_3 = Reimann_function(f, 0, 1, 0.0000001)
print(cbind(my_calculation_1, my_calculation_2, my_calculation_3))
```

```
##      my_calculation_1 my_calculation_2 my_calculation_3
## [1,]        0.8564276        0.8556324        0.8556245
```

```r
# Function in R
real_value = integrate(f, 0, 1)
difference = my_calculation_3 - real_value[[1]]
print(difference)
```

```
## [1] 8.032646e-08
```

```r
print(difference< (10^(-6)))
```

```
## [1] TRUE
```

Based on the results above, we can see that smaller width leads to higher accuracy while lower efficiency.

Problem 7

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.

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

For a refresher on Newton's method:
https://en.wikibooks.org/wiki/Calculus/Newton%27s_Method

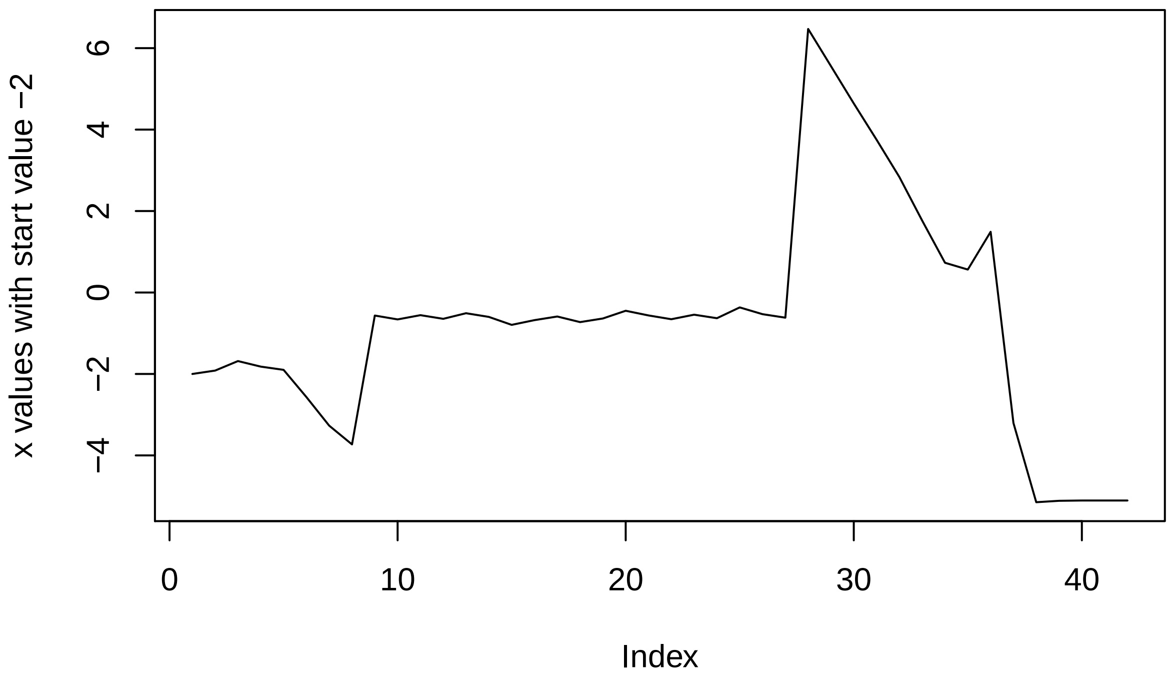
```r
# f(x)
f = function(x){
  f = 3^x - sin(x) + cos(5*x)
  return(f)
}
# f'(x)
f_derivative <- function(x) {
    fp = 3^x*log(3) - cos(x) - 5*sin(5*x)
    return(fp)
}
Root_Newton = function(f,f_derivative, start_val, tol){
  tmp = start_val
  x = tmp
  err = 1
  while(err > tol){
    x = x - f(x)/(f_derivative(x))
    tmp = c(tmp,x)
    err = abs(f(x) - 0)
  }
  return(list(x,tmp))
}
# Here interval: x belongs to (-10,10)
# Start_val = -2
y_1 = Root_Newton(f, f_derivative,-2, 0.000001)
root_1 = y_1[[1]]
print(root_1)
```

```
## [1] -5.107437
```

```r
# Check root
print(f(root_1))
```

```
## [1] -6.190048e-12
```

```r
plot(y_1[[2]], type = 'l', ylab = 'x values with start value -2')
```



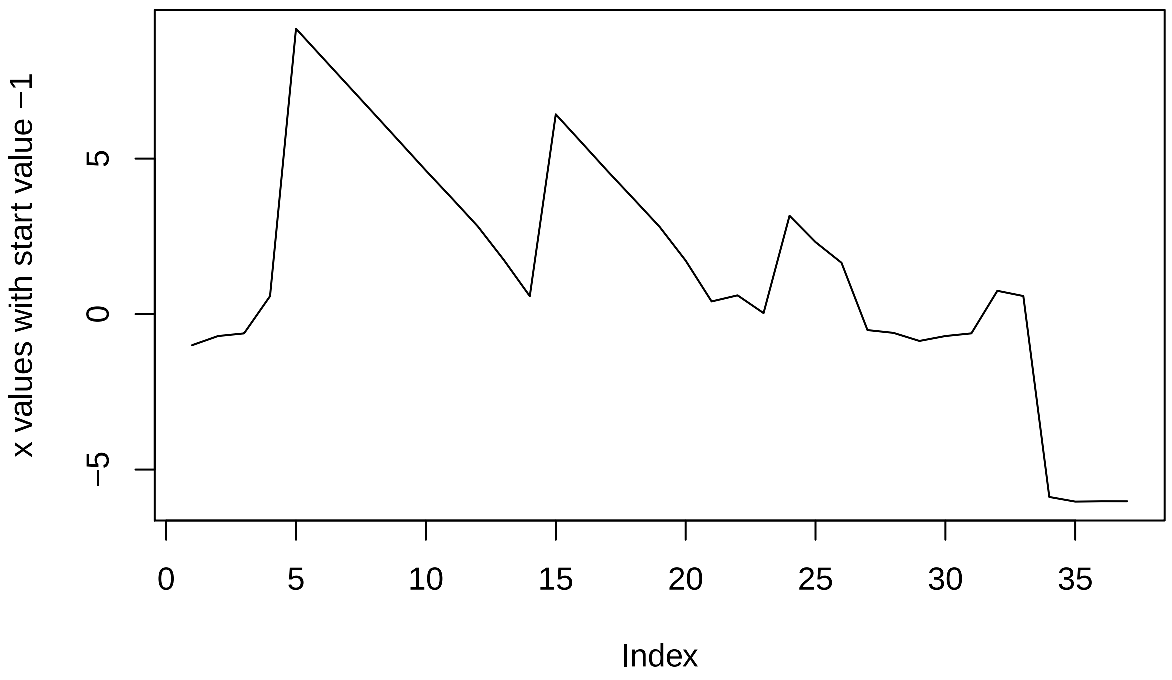
```r
# Start_val = -1
y_2 = Root_Newton(f, f_derivative,-1, 0.000001)
root_2 = y_2[[1]]
print(root_2)
```

```
## [1] -6.021155
```

```
# Check root
print(f(root_2))
```

```
## [1] -9.454361e-09
```

```
plot(y_2[[2]], type = 'l', ylab = 'x values with start value -1')
```



Problem 8

In most of your classes, you will be concerned with "sums of squares" of various flavors. SST = SSR + SSE for instance. Sums of square total (SST) = sums of square regression (SSR) + sums of square error (SSE). In this problem, we want to compare use of a for loop to using matrix operations in calculating these sums of squares. We will use data simulated using:

Without going too far into the Linear Regression material, we want to calculate SST =

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

Please calculate this using:

a. accumulating values in a for loop

b. matrix operations only

Note, you can precalculate mean(y) and create any vectors you need, ie perhaps a vector of 1 (ones). In both cases, wrap the calculation in the microbenchmark function. Report the final number and timings.

```
#X <- cbind(rep(1,100), rep.int(1:10, time=10))
#beta <- c(4,5)
sst_calculation = function(beta, X){
  y <- X%*%beta + rnorm(100)
  y_bar = matrix(mean(y), length(y), 1)
  start_time = Sys.time()
  sst_loop = 0
for (i in 1:length(y)){
  sst_loop = sst_loop + (y[i] - mean(y))^2
}
  end_time = Sys.time()
  time_cost_loop = end_time - start_time
  # matrix calculation
  start_time = Sys.time()
  sst_matrix = t(y - y_bar)%*%(y - y_bar)
  end_time = Sys.time()
  time_cost_matrix = end_time - start_time
  return(list(sst_loop, time_cost_loop, sst_matrix,time_cost_matrix ))
}
# Call function
X <- cbind(rep(1,100), rep.int(1:10, time=10))
beta <- c(4,5)
set.seed(0128)
results = sst_calculation(beta, X)
sst_loop = c(results[[1]], results[[2]])
sst_matrix = c(results[[3]], results[[4]])
result = as.data.frame(rbind(sst_loop, sst_matrix))
colnames(result) = c('SST', 'Time')
print(result)
```

```
##                 SST         Time
## sst_loop   21236.24 3.099442e-04
## sst_matrix 21236.24 1.621246e-05
```

Problem 9

Finish this homework by pushing your changes to your repo.

Only submit the .Rmd and .pdf solution files. Names should be formatted HW3_pid.Rmd and HW3_pid.pdf