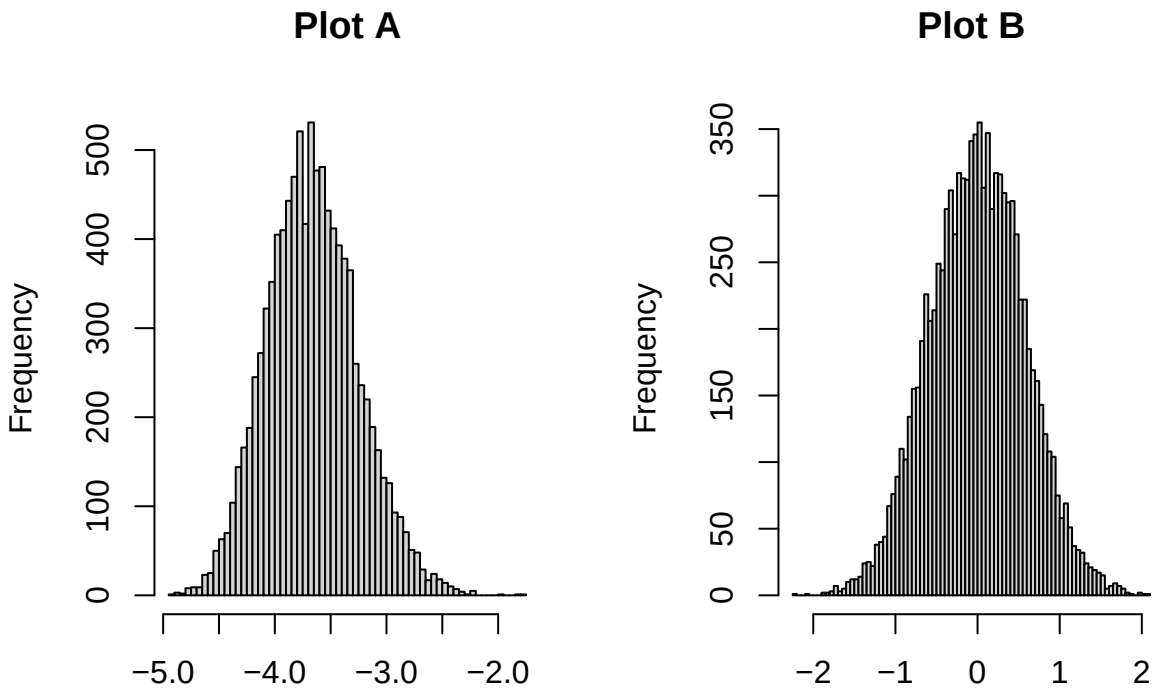
HW3 (105 points): DUE 11/5/2021

STAT 131A Fall 2021

Question 1 (5 points)

Consider the question of comparing the difference between two groups. We decide to use the difference in the medians to evaluate whether there is any significant difference between the two groups. One of the histograms below is the histogram of difference in the medians from the permutations in the permutation test comparing these two groups. The other histogram is the histogram of the difference in the medians from bootstrap resampling of the data to create confidence intervals. Which is which? (explain your reasoning)

```
blindData<-read.table("blindData.txt",sep="\t",header=FALSE)
par(mfrow=c(1,2))
hist(blindData[,1],breaks=100,main="Plot A",xlab="")
hist(blindData[,2],breaks=100,main="Plot B",xlab="")
```



My answer:
Plot A is the result of bootstrap resampling and plot B is the result of permutation test.

The reason is that the bootstrap statistics will center around the mean of the difference in the medians and the permutation statistics will center around zero if there aren't significant between two groups.

Question 2 (10 points)

Assume you have a data.frame, `df` with a column `y` and a column `group`. `y` is the response variable, and `group` is a factor variable that gives which group the observation is in; there are two possible groups labeled `group1` and `group2` (i.e. levels of the factor).

Consider the two following functions that 1) create a *single* bootstrap sample when given `df` 2) applies the function `FUN` to the bootstrap sample 3) returns the result of `FUN`. `FUN` can be any statistic that takes the data from two groups.

```
bootFun1<-function(df, FUN){
    group1<-df$y[df$group=="group1"]
    group2<-df$y[df$group=="group2"]
    sample1 <- sample(x=group1, size=length(group1),replace = TRUE)
    sample2 <- sample(x=group2, size=length(group2),replace = TRUE)
    return(FUN(sample1,sample2))
}
bootFun2<-function(df, FUN){
    whObs <- sample(x=1:nrow(df), size=nrow(df),replace = TRUE)
    sampleDf<-df[whObs,]
    sample1<-sampleDf$y[sampleDf$group=="group1"]
    sample2<-sampleDf$y[sampleDf$group=="group2"]
    return(FUN(sample1,sample2))
}
```

Describe the difference in the two strategies for re-sampling from the data.

My answer:
The first function first divide the df into 2 groups and re-sample the data respectively from two group, while the second function first mixed up the 2 groups into one pool and re-sample from the pool.
For the first function, every row in group 1 has a probability of '1 / length(group1)' to be chosen into sample1 and every row in group 2 has a probability of '1 / length(group2)' to be chosen into sample2, while every row in the 'df' has the same probability of '1 / (length(group1) + length(group2))' to be chosen.

Hint: You will likely want to create a simple toy dataset to try this out and play around with the code in each one.

Question 3

Consider the bootstrap function from the text (available on the website) for getting bootstrap confidence intervals for the results of fitting a line to the data using squared error.

```
bootstrapLM <- function(y,x, repetitions, confidence.level=0.95){
  stat.obs <- coef(lm(y~x))
  bootFun<-function(){
      sampled <- sample(1:length(y), size=length(y),replace = TRUE)
      coef(lm(y[sampled]~x[sampled]))
  }
```

```
  stat.boot<-replicate(repetitions,bootFun())
  # this next line that defines `nm` is advanced code, but
  # simply finds the name of the input x value
  # and saves this name as `nm`.
  nm <-deparse(substitute(x))

  row.names(stat.boot)[2]<-nm
  level<-1-confidence.level
  confidence.interval <- apply(stat.boot,1,quantile,probs=c(level/2,1-level/2))  ### *** ###
  out<-cbind("lower"=confidence.interval[1,],"estimate"=stat.obs,"upper"=confidence.interval[2,])
  return(list(confidence.interval = out, bootStats=stat.boot))
}
```

a (5 points) Explain what the function `bootFun` does. Specifically, what does the two lines of code within `bootFun` do? And what type of output does `bootFun` return? (a character string? A number? A vector? A list? A matrix?)

My answer:
1. The first line uses funtion 'sample' to generate random samples with replacement from the indices of y, with the same size of 'y'.
2. The second line funtion 'coef' to generate the estimated coefficients of a linear regression model from the sample.
3. `bootFun` return a vector at the length of 2.

b (5 points) Explain what type of object `stat.boot` is (a character string? A number? A vector? A list? A matrix?) and describe what are its actual entries

My answer:
`stat.boot` is a matrix of 2 rows and 'repetition' columns. Every column represent a repetition and the first index of a column is the bootstrapped intercept while the second index of a column is the bootstrapped slope.

c (5 points) Explain what the line of the code marked with *** does (marked above).

My answer:
This line use function 'apply' to calculate the `level/2` and `1-level/2` quantile for the intercept and slope of the 'stat.boot' respectively, where level is (1 - confidence level), therefore the 'apply' function return the bootstrap CIs of the intercept and slope.

Question 4

In the file `fitbit.csv` we have information from a fitbit device for a single person over the year. A fitbit device is something someone wears that records information regarding her activites throughout the day: her level of activity, the number of calories burned, the amount slept, etc (the company's website is www.fitbit.com).

The data that we have here is summarize per day, so that each entry of the data corresponds to a particular date.

The following code should read in the data and print out data from the first 5 rows of the first 8 columns.

```
fitbit<-read.csv("fitbit.csv",header=TRUE)
head(fitbit[,1:8])
```

```
##         Date caloriesBurned  steps distance plans minutesOfSedentaryActivities
## 1 01-01-2016          2.992 10.460     7,92     0                           685
## 2 01-03-2016          3.117 11.618     8,66    12                           776
## 3 01-04-2016          2.814 11.130     8,61     8                           900
## 4 01-05-2016          3.331 14.262     10,6     5                           666
## 5 01-06-2015          3.354 16.836    12,51     8                           586
## 6 01-06-2015          3.354 16.836    12,51     8                           586
##   minutesOfLightActivity minutesOfModerateActivity
## 1                    375                         0
## 2                    217                        59
## 3                    162                         3
## 4                    361                        21
## 5                    248                        35
## 6                    248                        35
```

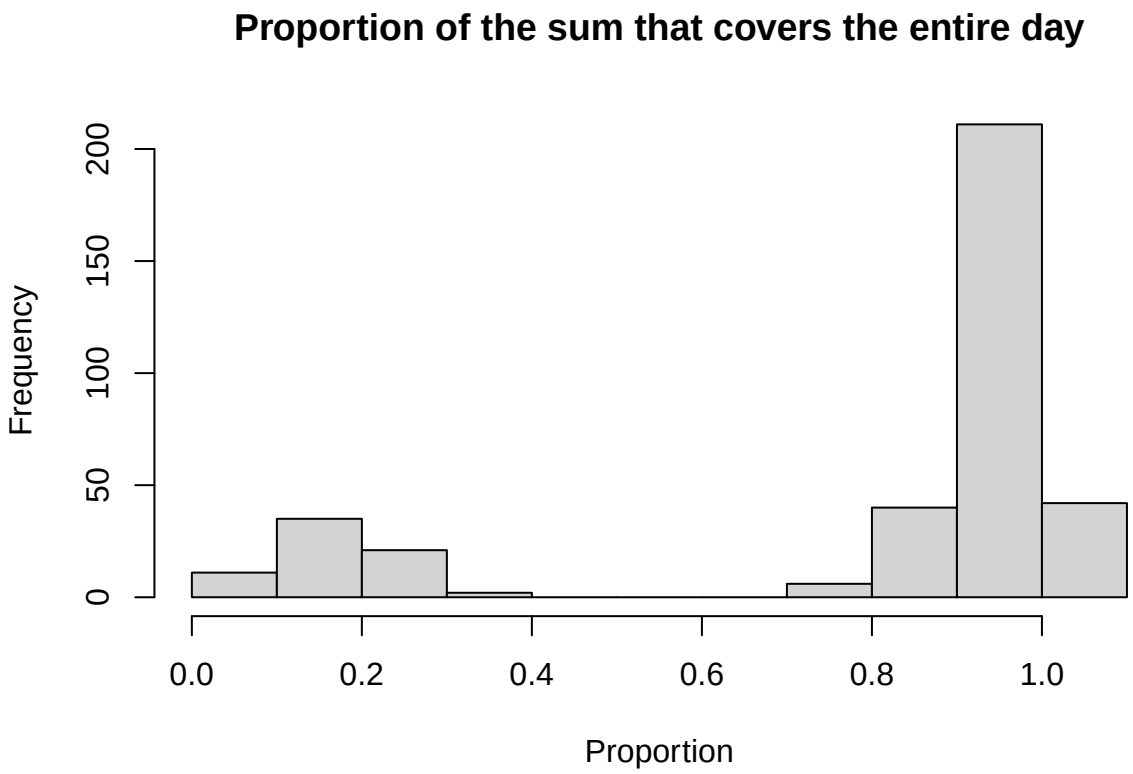
a (5 points) Many of the variables in this data set quantify the number of minutes spent doing activities, including sleeping. In principle, we might assume that these minutes should cover the whole day. However, the device might have been turned off or taken off at some point.

Let's consider this question. Look at the following variables: `MinutesOfSleep`, `minutesOfLightActivity`, `minutesOfModerateActivity`, `minutesOfSedentaryActivities`, and `minutesOfIntenseActivity`. How well do these sum up to cover the entire day?

```
# code to evaluate the total number of minutes
totMinutes <- with(fitbit, (MinutesOfSleep + minutesOfLightActivity
                            + minutesOfModerateActivity + minutesOfSedentaryActivities
                            + minutesOfIntenseActivity))
proportion <- totMinutes/(24 * 60)
summary(proportion)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0010  0.8693  0.9427  0.8020  0.9773  1.0424
```

```
hist(proportion, main = 'Proportion of the sum that covers the entire day',
     xlab = 'Proportion')
```



My answer:
We can see from the summary that the mean of the proportion that the total minutes cover a entire day is 0.9427, which means the total minutes can actually cover an entire day in many cases. However, we can see from the histogram that there are some days that the total minutes are totally below the minutes of an entire day, which means it can't interpret an day well. There are also some cases that the total minutes exceed the minutes of an entire day.

b (5 points) Going forward, we are going to consider the question of whether the amount of sedentary activity is predictive of the minutes of sleep per night. We are going to first normalize our data, so that both of these quantities are out of the total amount of minutes recorded in the day with the device. Calculate the total number of minutes recorded each day in the five variables from part a (call it `totMinutes`), and make new variables `pctAsleep` and `pctSedentary` that are the percentage of all tracked minutes. Print out a `summary` of each of these two variables to show that it worked.

```
#code to make new variables as percent of total minutes.
pctAsleep <- fitbit$MinutesOfSleep / totMinutes
pctSedentary <- fitbit$minutesOfSedentaryActivities / totMinutes
summary(pctAsleep)
```

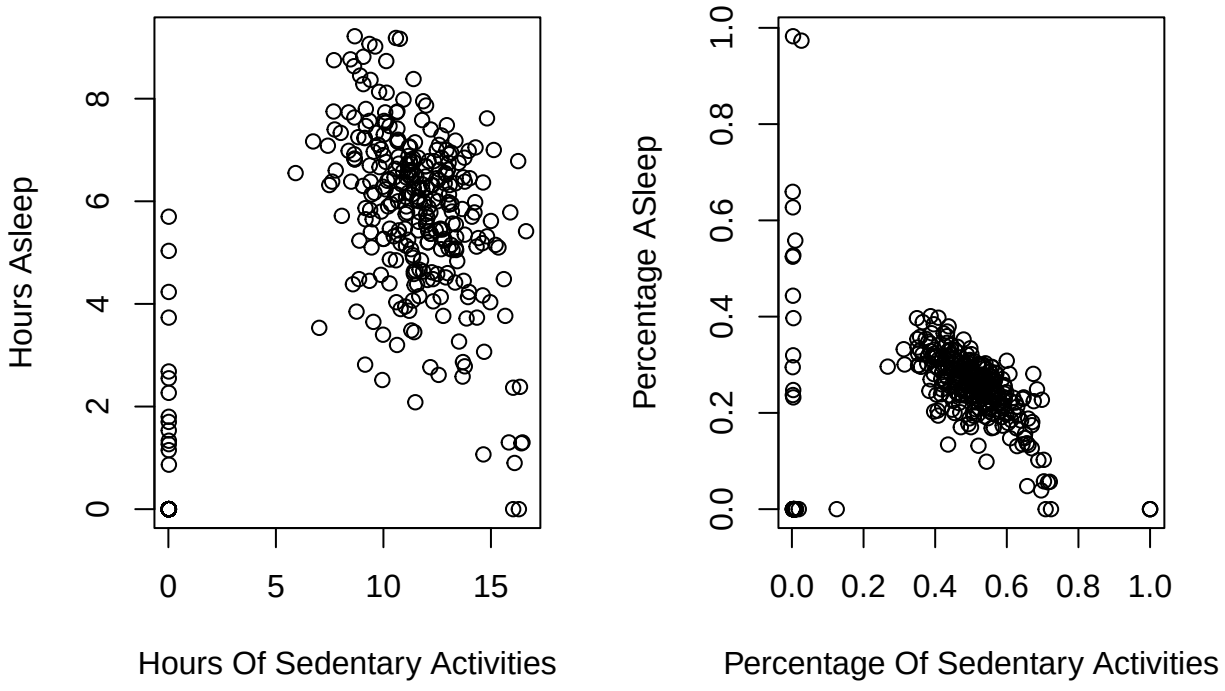
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1938  0.2489  0.2267  0.2935  0.9825
```

```
summary(pctSedentary)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.002275 0.396158 0.486914 0.422188 0.552060 1.000000
```

c (10 points) Make a plot of `MinutesAsleep` against `minutesOfSedentaryActivites` and another plot of `pctAsleep` against `pctSedentary`. For the minutes plot, convert the minutes into hours. Use `par(mfrow= ...)` to make the two plots side-by-side (or something else like patchwork if you are using ggplot), and label the two plots appropriately. Comment on the affect of converting them to percentages.

```
# code to plot the two plots side-by-side
par(mfrow = c(1, 2))
with(fitbit, plot(minutesOfSedentaryActivities/60, MinutesOfSleep/60,
 ylab = "Hours Asleep", xlab = "Hours Of Sedentary Activities"))
plot(pctSedentary, pctAsleep, ylab = "Percentage ASleep",
     xlab = "Percentage Of Sedentary Activities")
```



My answer:
By converting them to percentages, the relative percentage is more condensed and concentrated around the range of 0.2-0.4 at x and 0.3-0.7 at y and the plot shows a linear relation between the two percentages. However, the points in the hours' plot just scattering at a wide area which you are less possible to tell the linear trend.

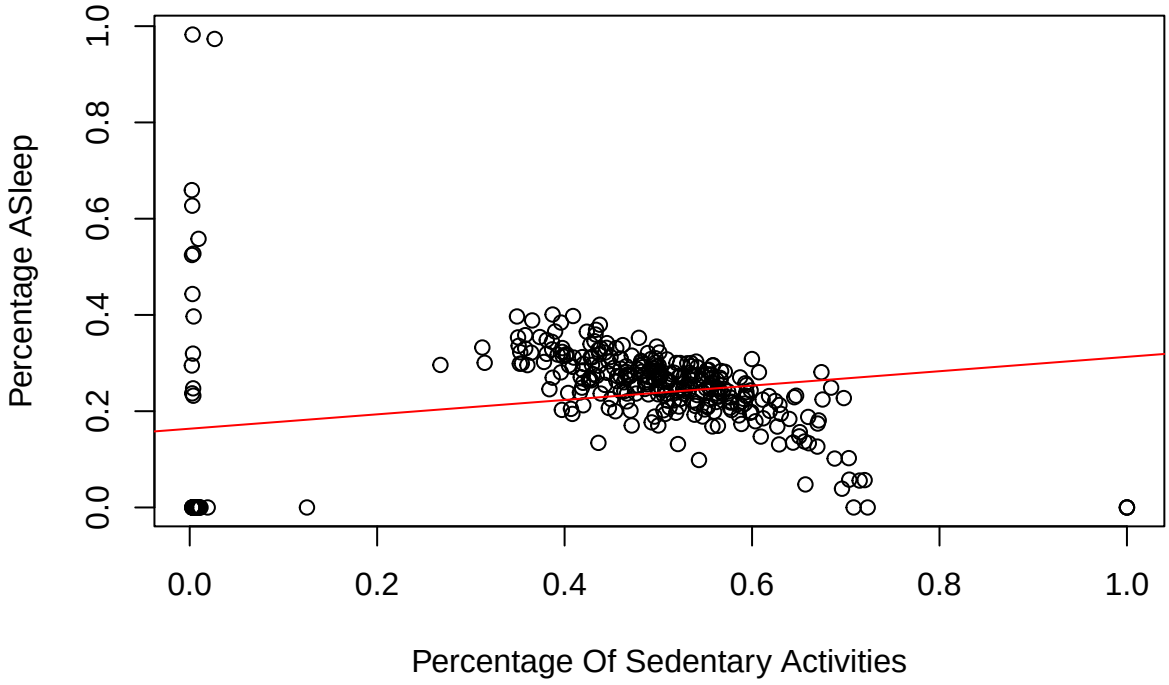
d (15 points) Plot the variable `pctAsleep` against `pctSedentary` like above, only now draw in the least squares regression line. Also print out the coefficients of the lines. Interpret the line and plots; be specific to this data – e.g. as you increase percent of time spent in sedentary minutes by 10%, how is `pctAsleep` predicted to change?

```
# Code here for plot with regression line
plot(pctSedentary, pctAsleep, ylab = "Percentage ASleep",
     xlab = "Percentage Of Sedentary Activities")
```

```r
coef <- coef(lm(pctAsleep~pctSedentary))
coef
```

```
##  (Intercept) pctSedentary
##    0.1636420    0.1492903
```

```r
abline(coef[1], coef[2], col = "red")
```



My answer:
The linear regression line shows that the percentage of Sedentary Activities has a positive correlation with the percentage Asleep, which means when the percentage of Sedentary Activities goes up, the percentage Asleep will go up consequently. To be specific, if you increase percent of time spent in sedentary minutes by 10%, `pctAsleep` is predicted to increase 1.49%.

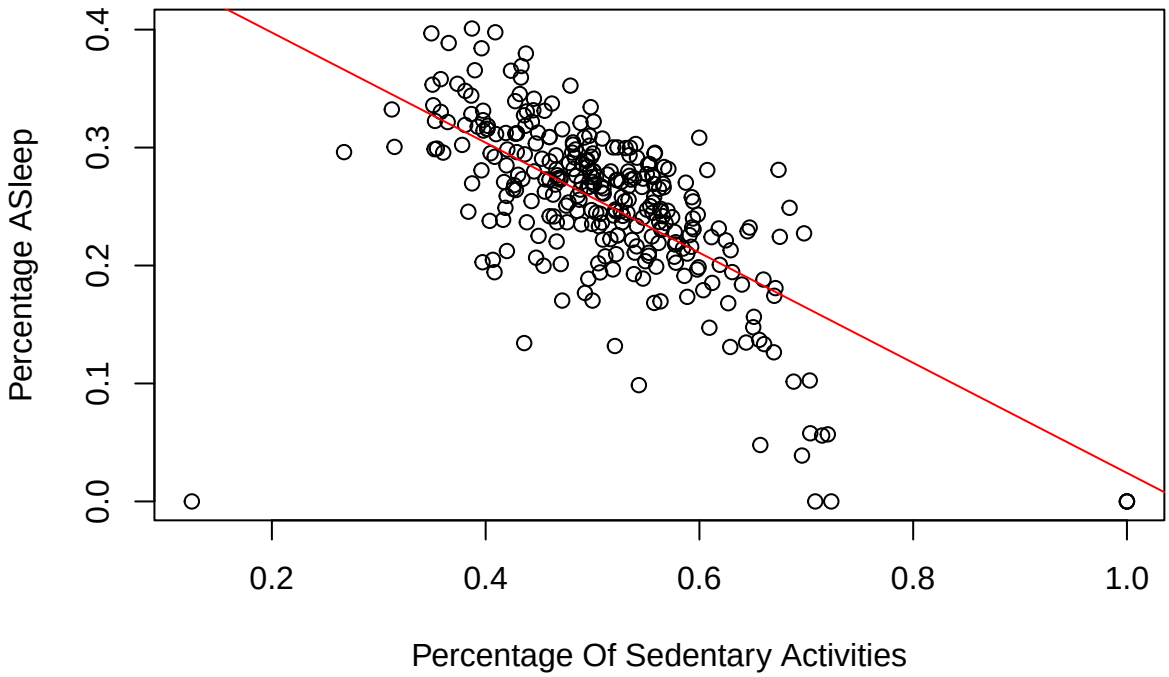
e (15 points) We might consider the effect of the (practically) zero levels of sedentary activity. Remove these points from the above analysis in d, and compare the results to that found in d, and draw conclusions as to the effect of these data points. What conclusion would you draw as to whether they should be included in the analysis?

```r
# Code here for plot with regression line removing zero sedentary
plot(pctSedentary[pctSedentary > 0.05], pctAsleep[pctSedentary > 0.05],
 ylab = "Percentage ASleep", xlab = "Percentage Of Sedentary Activities")
coef.removeZero <- coef(lm(pctAsleep[pctSedentary > 0.05]~pctSedentary[pctSedentary > 0.05]))
coef.removeZero
```

```
##                       (Intercept) pctSedentary[pctSedentary > 0.05]
```

```
##                          0.4908497                          -0.4666984
```

```
abline(coef.removeZero[1], coef.removeZero[2], col = "red")
```



My answer:
After remove the zero level data, the prediction of the linear regression model seems more reasonable.
From the relative percentage plot, we can tell that there are some outliers where 'pctSedentary' are close to zero (less than 0.05%) and these outliers obviously affect the prediction of the linear regression model. The slope of the linear regression model should be negative since the data show a negative correlation, but the original linear regression model has a positive slope. After remove the zero level data, the estimated linear regression model predicts the trend of the data better.

f (20 points) Create bootstrap confidence intervals as well as parametric confidence intervals for the coefficients of the regression you found in e. (you can use the function from Question 3). Plot the resulting confidence intervals and compare them.

```
# Code here bootstrap and parametric confidence intervals
#bootstrap CI
bootstrap.LM <- bootstrapLM(pctAsleep[pctSedentary > 0.05],
                            pctSedentary[pctSedentary > 0.05],
                            repetitions = 1000)
bootstrap.CI <- bootstrap.LM$confidence.interval
#parametric CI
parametric.CI <- confint(lm(pctAsleep[pctSedentary > 0.05]~pctSedentary[pctSedentary > 0.05]))
```

```
bootstrap.CI
```

```
##                                       lower   estimate      upper
## (Intercept)                       0.4157446  0.4908497  0.5434488
## pctSedentary[pctSedentary > 0.05] -0.5656776 -0.4666984 -0.3226191
```

```
parametric.CI
```

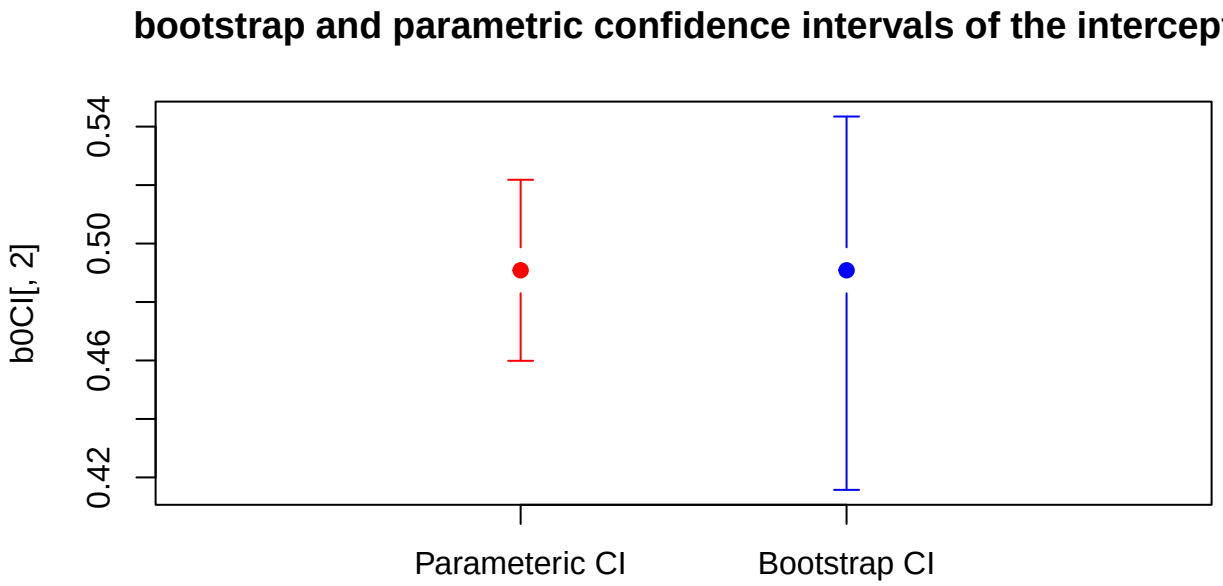
```
##                                       2.5 %     97.5 %
## (Intercept)                       0.4598885  0.5218110
## pctSedentary[pctSedentary > 0.05] -0.5261015 -0.4072953
```

```
#plot
b0CI<-rbind(c(lower=parametric.CI[1,1], estimate=unname(coef.removeZero[1]),
              upper=parametric.CI[1,2]), bootstrap.CI[1,])
b1CI<-rbind(c(lower=parametric.CI[2,1], estimate=unname(coef.removeZero[2]),
              upper=parametric.CI[2,2]), bootstrap.CI[2,])
library(gplots)
```

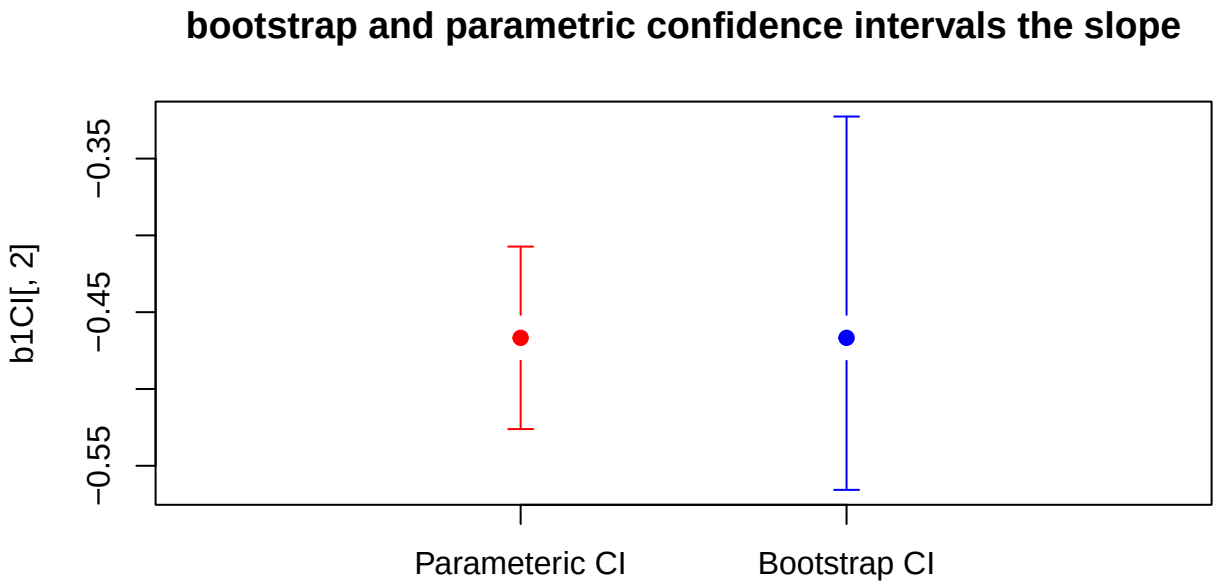
```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
par(mar=c(8,4,4,1))
plotCI(b0CI[,2], ui=b0CI[,3], li=b0CI[,1],
       main="bootstrap and parametric confidence intervals of the intercept",
       col=c("red", "blue"),pch=19,xaxt="n",xlim=c(0,3))
axis(1,at=c(1,2),labels=c("Parameteric CI","Bootstrap CI"),las=1)
```




```
par(mar=c(8,4,4,1))
plotCI(b1CI[,2],ui=b1CI[,3] ,li=b1CI[,1],
       main="bootstrap and parametric confidence intervals the slope",
       col=c("red", "blue"), pch=19, xaxt="n", xlim=c(0,3))
axis(1,at=c(1,2),labels=c("Parameteric CI","Bootstrap CI"),las=1)
```




```
b0CI
```

```
##          lower  estimate     upper
## [1,] 0.4598885 0.4908497 0.5218110
## [2,] 0.4157446 0.4908497 0.5434488
```

```
b1CI
```

```
##           lower   estimate      upper
## [1,] -0.5261015 -0.4666984 -0.4072953
## [2,] -0.5656776 -0.4666984 -0.3226191
```

My answer:
The bootstrap CI of the intercept is (0.46, 0.52) and the parametric CI of the intercept is (0.42, 0.54), which are quite similar, but the bootstrap CI is wider than the parametric CI, shows that we are more confident on the predicted results.
The bootstrap CI of the slope is (-0.53, -0.41) and the parametric CI of the intercept is (-0.57, -0.34). The same as the CIs of the intercept, the CIs for bootstrap and parametric are close, but the bootstrap CI is a little bit wider.

g (5 points) We've seen that we do not always have all of the minutes of a day recorded. What affect could these missing minutes have on our above analysis? Be specific. A good idea is to first try to think of extreme examples of what could be happening in those minutes that might change your conclusion, so you can think about why not having that data might be a problem for your interpretation of the relationship between of sedentary activity and sleep levels. Then try to think of the best case scenario where lacking your data would not affect your results. Then return to the actual data (i.e. real life) and think how plausible it is to be worried about this problem.

My answer:
Consider the case that the missing minutes are missing for a specific event instead of randomly missing at different event. This could affect the data and lead to a bad and misleading prediction about the regression model. The best case scenario is that the missing minutes are just randomly missing and distribute equally among different events so that it won't affect the predicted model that much.
However, in the real life case, it is less likely that the missing minutes will distribute randomly among different activities, so the linear regression model predicted above could be wrong and biased from the real case.