R programming and bootstrapping

50 marks

In this question, you will write a couple of simple `S3` methods and then use them to get a *bootstrap sample* of any attribute values using the `Map()` function. You will also be writing a `bootstrap()` function in two different ways: first using `Map()` then using `sapply()`.

The data set `cars` will be used to illustrate the differences and for statistical intepretation of the bootstrap distribution.

a. **(4 marks)** One minor annoyance with `R` is that there is not a single function that will retrieve the number of population units for any data representation. For example, the data frame `cars` has 50 units, as does each of its variables `cars$dist` and `car$speed$. A simple list`l <- list(a = "apple", b = "banana", c = "cucumber")‘ has three units (presumably from a population of types of fruit).

   i. *(2 marks)* To get around this problem, write an `S3` generic function called `n_units()` that will return the correct number of units. You need complete the following definition for the generic function

```
n_units <- function(x) {}
```

   as well as methods specialized for each of `data.frame` and `matrix`.

   The default method you write should work for the rest.

```
n_units <- function(x){
  UseMethod("n_units", x)
}
n_units.default <- function(x){
  length(x)
}
n_units.data.frame <- function(x){
  dim(x)[1]
}
n_units.matrix <- function(x){
  ncol(x)*nrow(x)
}
```

   ii. *(2 marks)* Show the results of the following uses of your function: Cars: Dataframe cars$speed: double HairEyeColor: double lm(dist ~ speed, data = cars): List (intercept, speed)

```
n_units(cars)
```

```
## [1] 50
```

```
n_units(cars$speed)
```

```
## [1] 50
```

```
n_units(HairEyeColor)
```

```
## [1] 32
```

```
n_units(lm(dist ~ speed, data = cars))
```

```
## [1] 12
```

```
n_units(n_units)
```

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

b. **(5 marks)** Similarly, when it comes to extracting the values for specified units, there is no single convenient function to do the job. Which function you use depends on whether you are extracting values from a vector, a data frame, or a matrix.

   i. *(3 marks)* To get around this problem, write an `S3` generic function called `getValues()` that will return the values of `x` specified by `indices`. You need complete the following generic function definition

   ```
   getValues <- function(x, indices) { ... }
   ```

   as well as write methods specialized for each of `data.frame` and `matrix`, and `list`.

   The default method should work for anything else.

   For matrices and data frames the values returned should preserve that structure.

```
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
getValues <- function(x, indices) {
  UseMethod("getValues", x)
}
getValues.default <- function(x, indices){
  x[indices]
}
getValues.list <- function(x, indices){
  x[[indices]]
}
getValues.data.frame <- function(x, indices){
  x[indices,]
}
getValues.matrix <- function(x, indices){
  x[indices,]
}
```

   ii. *(2 marks)* Show the results of the following uses of your function:

```
getValues(mtcars, c("Pontiac Firebird"))
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Pontiac Firebird 19.2   8  400 175 3.08 3.845 17.05  0  0    3    2
```

```
getValues(data.frame(alphabet = LETTERS), 1:3)
```

```
## [1] "A" "B" "C"
```

```
getValues(HairEyeColor, 1:3)
```

```
## [1] 32 53 10
```

```
getValues(lm(dist ~ speed, data = cars), 1)
```

```
## $coefficients
## (Intercept)       speed
##  -17.579095    3.932409
```

c. **(7 marks)** Functions that return functions.

   Imagine a function `fit(formula, data, ...)` that (somehow) fits a model given by `formula` to the data contained in `data`.

   Examples of potential functions `fit()` include

   - `lm(formula, data  = data, ...)`
   - `glm(formula, data = data, ...)`
   - `loess(formula, data = data, ...)`

   Of course, the variables in the `formula` would have to be present in `data`.

What we want here is a wrapper function that will take the `fit` function and the `formula` as arguments (plus any other arguments `fit` might use) and will return a function which takes only `data` as an argument but will calculate the `fit` on `data` using the `formula` and any other arguments `...` passed on to `fit`.

i. *(3 marks)* Create the function `gitFitFn()` as described above.

Show your code.

```
getFitFn <- function(fit, formula, ...){
  myFunc = function(data){
    fit(formula, data = data,...)
  }
  return(myFunc)
}
```

```
ii. *(2 marks)* Test your function `gitFitFn()` by getting `result1` and `result2` as above and evaluating

    Show your code

    (including construction of `myFit()` as well as of `result1` and `result2`.)
```

```
myFit <- getFitFn(fit = lm, formula = dist ~ speed)
result1 <- myFit(data = cars)
result2 <- lm(dist ~ speed, data = cars)
coef(result1) == coef(result2)
```

```
## (Intercept)       speed
##        TRUE        TRUE
```

```
iii. *(2 marks)* Now you will test that your function works with additional arguments for `fit`.

     Repeat the above test except now pass the additional argument  `model = FALSE`, first to  `getFitFn()`

     Reconstruct both `result1` and `result2` with `model = FALSE`.

     Test the equality of the coefficients from each result.

     Show your code.
```

```
myFit <- getFitFn(fit = lm, formula = dist ~ speed, model = FALSE)
result1 <- myFit(data = cars)
result2 <- lm(dist ~ speed, data = cars, model = FALSE)
coef(result1) == coef(result2)
```

```
## (Intercept)       speed
##        TRUE        TRUE
```

d. **(8 marks) A bootstrap distribution.**

For any sample $\mathcal{S}$ of size n selected from some population $\mathcal{P}$ (using simple random sampling), the *sampling distribution* of any attribute $a(\mathcal{S})$ is used to make inferences about $a(\mathcal{P})$. Unfortunately, we typically do not know that distribution.

However, we might estimate it by mimicking the sampling plan using the observed sample $\mathcal{S}$ in place of the unavailable population $\mathcal{P}$. That is, samples $\mathcal{S}_1^*, \mathcal{S}_2^*, \ldots, \mathcal{S}_B^*$ of size n are randomly selected (this time **with** replacement) from $\mathcal{S}$ in place of $\mathcal{P}$. There will be n^n possible samples but typically only $B << n^n$ bootstrap samples $\mathcal{S}_i^*$ are selected at random.

The *bootstrap distribution* of $a(\mathcal{S})$ is then estimated by the collection of values $a(\mathcal{S}_1^*), a(\mathcal{S}_2^*), \ldots, a(\mathcal{S}_B^*)$.

In this question you will write and use a function that creates this estimated bootstrap distribution.

i. *(5 marks)* Implement the following function using the `Map()` function as well as any of the functions from earlier parts of this question which you might need.

```
bootstrap <- function(data, attribute, B = 1000) { ... }
```

Here

- `data` is the data set containing the sample $\mathcal{S}$ and the values on all variates of interest,
- `attribute` is a function that calulates the attribute value when called on `data` (or a bootstrap sample from `data` of the same size), and
- `B` is the number of bootstrap samples to be used in estimating the bootstrap distribution.

`bootstrap` **returns** the result of the `Map()` call as the collection values of the attribute evaluated on each of the B bootstrap samples $a(\mathcal{S}_1^*), a(\mathcal{S}_2^*), \ldots, a(\mathcal{S}_B^*)$.

Show your code.

```
bootstrap <- function(data, attribute, B = 1000) {
  bootSamp <- lapply(1:B,
                     FUN = function(p){
                       samp_index <- sample(1:n_units(data), n_units(data), replace = TRUE)
                       samp_vals <- getValues(data, samp_index)
                     })
  Map(attribute, bootSamp)
}
```

ii. *(3 marks)* Test your bootstrap function by getting the following data:

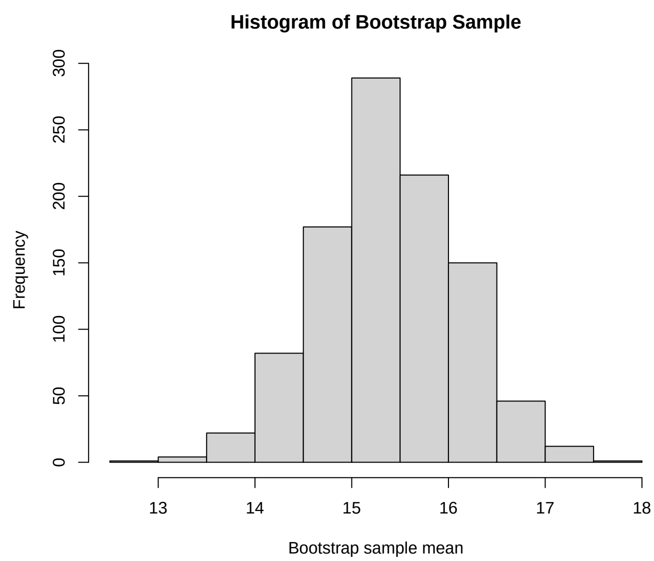
```
set.seed(314159)
bSamples <- bootstrap(cars$speed, mean)
```

And drawing a histogram of the resulting values.

Make sure the histogram has a meaningful title and labels.

Show your code.

```
set.seed(314159)
bSamples <- bootstrap(cars$speed, mean)
hist(unlist(bSamples),xlab = "Bootstrap sample mean", main = "Histogram of Bootstrap Sample")
```



e. **(6 marks)** Use the functions you constructed above to estimate the bootstrap distribution the quadratic coefficient estimator when fitting the stopping distance as a quadratic function of the car's speed to the data set `cars`.

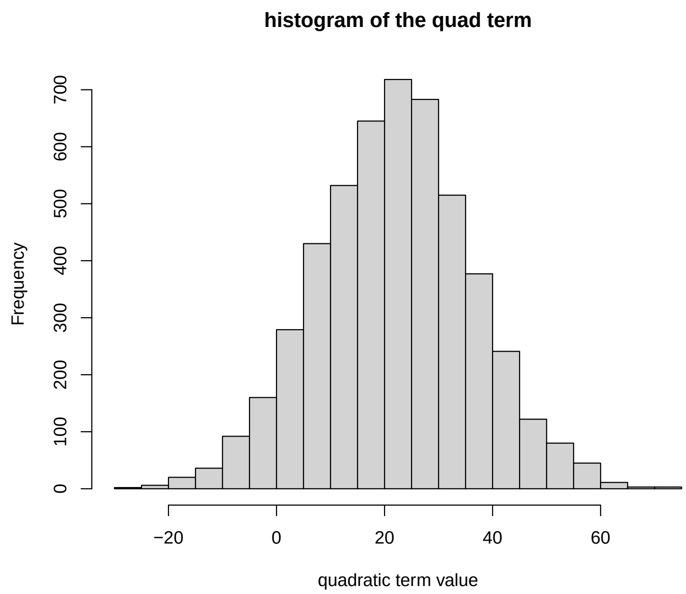
Use `B = 5000` and draw a histogram of the estimated bootstrap distribution of the slope estimator.

Make sure the histogram has meaningful labels, title, and legend.

Show all your code. **AGAIN** `set.seed(31459)` **before** calling `bootstrap()`.

What do you conclude about the contribution of the quadratic term from this histogram?

```
set.seed(314159)
quad_fit <- getFitFn(fit = lm, formula = dist ~ poly(speed,2))
bsamples <- bootstrap(cars, quad_fit, B = 5000)
#hist(unlist(bsamples))
quad_list <- rep(NA, 5000)
for (i in 1:5000){
  quad_list[i]<-bsamples[[i]]$coefficients[[3]]
}
hist(quad_list, breaks = 25, main = "histogram of the quad term", xlab = "quadratic term value")
```



f. **(10 marks)** The `bootstrap()` function defined earlier always returned a list. However in plotting, data.frames or numeric vectors are generally preferred.

Consider again the bootstrap distribution for the coefficients fitting a model of `dist` as a quadratic function of `speed`. Here the difference between `Map()` and `sapply()` is investigated in the context of implementing the `bootstrap()` function.

i. *(1 mark)* Using the `bootstrap()` function (implemented previously using `Map()`) get a bootstrap sample of all three coefficients produced by the fit.

In this case,

- `set.seed(314159)` again
- use `B = 50`
- assign the result to `bsamples <- bootstrap(...)`
- show the coefficients for the first bootstrap sample generated

Show your code.

```
bootstrap <- function(data, attribute, B = 50) {
  bootSamp <- lapply(1:B,
                     FUN = function(p){
                       samp_index <- sample(1:n_units(data), n_units(data), replace = TRUE)
                       samp_vals <- getValues(data, samp_index)
                     })
```

```
  Map(attribute, bootSamp)
}
set.seed(314159)
quad_fit <- getFitFn(fit = lm, formula = dist ~ poly(speed,2))
bSamples <- bootstrap(cars, quad_fit, B = 50)
bSamples[[1]]
```

```
##
## Call:
## fit(formula = formula, data = data)
##
## Coefficients:
##     (Intercept)  poly(speed, 2)1  poly(speed, 2)2
##           45.92           172.93            48.30
```

ii. *(3 marks)* Ideally, `bSamples` from part (i) would be a `data.frame` with three variables, instead of a list of vectors, each of length 3. Perhaps if `sapply()` were used in place of `Map()` in `bootstrap()`, a simpler structure for `bSamples` might come out.

Rewrite your bootstrap function so that it uses (and returns the result of) `sapply()` instead of `Map()`.

Use this rewritten `bootstrap()` function to recreate `bSamples` as in part (i). (Again, `set.seed(314159)` first and use `B = 50`.)

- What is the class of `bSamples` now?
- Show the coefficient estimates from the first bootstrap sample.

Show your code.

`Answer:   The class of bSamples is matrix, array now`

```
bootstrap <- function(data, attribute, B = 50) {
  bootSamp <- lapply(1:B,
                     FUN = function(p){
                       samp_index <- sample(1:n_units(data), n_units(data), replace = TRUE)
                       samp_vals <- getValues(data, samp_index)
                     })
  sapply(bootSamp, attribute)
}
set.seed(314159)
quad_fit <- getFitFn(fit = lm, formula = dist ~ poly(speed,2))
bSamples <- bootstrap(cars, quad_fit, B = 50)
class(bSamples)
```

```
## [1] "matrix" "array"
```

```
bSamples[[1]]
```

```
##     (Intercept) poly(speed, 2)1 poly(speed, 2)2
##         45.9200        172.9296         48.2995
```

iii. *(6 marks)* Instead of returning only the coefficients, define an attribute that is the whole fit. Using the `bootstrap()` function of part (ii), generate the bootstrap sample of **fits** (Again, `set.seed(314159)` first and use `B = 50`.)

- Explain why `bSamples` is a `matrix`?

  bSamples is a matrix in this case is because that we change from Map to sapply, and sapply will return a matrix

- What class is a row of `bSamples`?

  Class of row of 'bSamples' is list

- What class is a column of `bSamples`?

  Class of column of 'bSamples' is list

- Show **3 distinctly different** ways to extract the coefficients of the fit from the first bootstrap sample.

Show your code.

```
set.seed(314159)
fit <- getFitFn(fit = lm, formula = dist ~ poly(speed,2))
bSamples <- bootstrap(cars, fit, B = 50)
bSamples[,1]
```

```
## $coefficients
##     (Intercept) poly(speed, 2)1 poly(speed, 2)2
##         45.9200        172.9296         48.2995
##
## $residuals
##          10          34          16          20          45          17
##  -2.3885346  27.4176424   0.5858833  -3.1215608 -29.3281495   8.5858833
##          15        17.1          49          14          35          50
##   5.8302254   8.5858833  28.3333860   1.8302254  35.4176424 -15.4681805
##          13        13.1        45.1        13.2          30          42
##  -2.1697746  -2.1697746 -29.3281495  -2.1697746  -3.0225054  -5.0913683
##          40        50.1        20.1          37        49.1        37.1
## -13.0913683 -15.4681805  -3.1215608  -8.6053119  28.3333860  -8.6053119
##          38        49.2        10.1          41        41.1           7
##  13.3946881  28.3333860  -2.3885346  -9.0913683  -9.0913683   0.9296034
##           4          28          36          43        28.1        49.3
##   9.1054052   2.0742449 -18.6053119   2.9086317   2.0742449  28.3333860
##        36.1        28.2        38.1          24          18        36.2
## -18.6053119   2.0742449  13.3946881 -13.2921070   8.5858833 -18.6053119
##        28.3        13.3           1        40.1           3        13.4
##   2.0742449  -2.1697746 -10.8867114 -13.0913683  -8.8945948  -2.1697746
##           9        42.1
##  16.9296034  -5.0913683
##
## $effects
##     (Intercept) poly(speed, 2)1 poly(speed, 2)2
##   -324.70343392   -172.92964155     48.29950149     -1.83937577    -33.96515527
##
##     10.65293991      8.70759029     10.65293991     23.16587607      4.70759029
##
##     33.81470854    -21.14075788      0.70759029      0.70759029    -33.96515527
##
##      0.70759029     -3.94231459     -7.98424109    -15.98424109    -21.14075788
##
##     -1.83937577    -10.86593363     23.16587607    -10.86593363     11.13406637
##
##     23.16587607      1.32457537    -11.98424109    -11.98424109      5.50389515
##
##     16.41586271      1.86299698    -20.86593363      0.01575891      1.86299698
##
##     23.16587607    -20.86593363      1.86299698     11.13406637    -12.76935675
##
##     10.65293991    -20.86593363      1.86299698      0.70759029     -0.61115741
##
##    -15.98424109     -1.58413729      0.70759029     21.50389515     -7.98424109
##
```

```
## $rank
## [1] 3
##
## $fitted.values
##        10        34        16        20        45        17        15      17.1
##  19.38853  48.58236  25.41412  29.12156  83.32815  25.41412  22.16977  25.41412
##        49        14        35        50        13      13.1      45.1      13.2
##  91.66661  22.16977  48.58236 100.46818  22.16977  22.16977  83.32815  22.16977
##        30        42        40      50.1      20.1        37      49.1      37.1
##  43.02251  61.09137  61.09137 100.46818  29.12156  54.60531  91.66661  54.60531
##        38      49.2      10.1        41      41.1         7         4        28
##  54.60531  91.66661  19.38853  61.09137  61.09137  17.07040  12.89459  37.92576
##        36        43      28.1      49.3      36.1      28.2      38.1        24
##  54.60531  61.09137  37.92576  91.66661  54.60531  37.92576  54.60531  33.29211
##        18      36.2      28.3      13.3         1      40.1         3      13.4
##  25.41412  54.60531  37.92576  22.16977  12.88671  61.09137  12.89459  22.16977
##         9      42.1
##  17.07040  61.09137
##
## $assign
## [1] 0 1 1
##
## $qr
## $qr
##      (Intercept) poly(speed, 2)1 poly(speed, 2)2
## 10    -7.0710678    2.775558e-17   -1.821460e-17
## 34     0.1414214   -1.000000e+00   -1.804112e-16
## 16     0.1414214   -7.693684e-02    1.000000e+00
## 20     0.1414214   -4.894419e-02    1.092184e-01
## 45     0.1414214    2.029897e-01   -1.353552e-01
## 17     0.1414214   -7.693684e-02    8.845262e-02
## 15     0.1414214   -1.049295e-01    5.809866e-02
## 17.1   0.1414214   -7.693684e-02    8.845262e-02
## 49     0.1414214    2.309823e-01   -2.104707e-01
## 14     0.1414214   -1.049295e-01    5.809866e-02
## 35     0.1414214    6.302642e-02    9.640038e-02
## 50     0.1414214    2.589750e-01   -2.951743e-01
## 13     0.1414214   -1.049295e-01    5.809866e-02
## 13.1   0.1414214   -1.049295e-01    5.809866e-02
## 45.1   0.1414214    2.029897e-01   -1.353552e-01
## 13.2   0.1414214   -1.049295e-01    5.809866e-02
## 30     0.1414214    3.503377e-02    1.139871e-01
## 42     0.1414214    1.190117e-01    3.246256e-02
## 40     0.1414214    1.190117e-01    3.246256e-02
## 50.1   0.1414214    2.589750e-01   -2.951743e-01
## 20.1   0.1414214   -4.894419e-02    1.092184e-01
## 37     0.1414214    9.101907e-02    6.922554e-02
## 49.1   0.1414214    2.309823e-01   -2.104707e-01
## 37.1   0.1414214    9.101907e-02    6.922554e-02
## 38     0.1414214    9.101907e-02    6.922554e-02
## 49.2   0.1414214    2.309823e-01   -2.104707e-01
## 10.1   0.1414214   -1.329221e-01    1.815658e-02
## 41     0.1414214    1.190117e-01    3.246256e-02
## 41.1   0.1414214    1.190117e-01    3.246256e-02
## 7      0.1414214   -1.609148e-01   -3.137364e-02
## 4      0.1414214   -2.448927e-01   -2.374931e-01
## 28     0.1414214    7.041116e-03    1.219857e-01
```

```
## 36     0.1414214    9.101907e-02    6.922554e-02
## 43     0.1414214    1.190117e-01    3.246256e-02
## 28.1   0.1414214    7.041116e-03    1.219857e-01
## 49.3   0.1414214    2.309823e-01   -2.104707e-01
## 36.1   0.1414214    9.101907e-02    6.922554e-02
## 28.2   0.1414214    7.041116e-03    1.219857e-01
## 38.1   0.1414214    9.101907e-02    6.922554e-02
## 24     0.1414214   -2.095154e-02    1.203961e-01
## 18     0.1414214   -7.693684e-02    8.845262e-02
## 36.2   0.1414214    9.101907e-02    6.922554e-02
## 28.3   0.1414214    7.041116e-03    1.219857e-01
## 13.3   0.1414214   -1.049295e-01    5.809866e-02
## 1      0.1414214   -3.288707e-01   -5.299057e-01
## 40.1   0.1414214    1.190117e-01    3.246256e-02
## 3      0.1414214   -2.448927e-01   -2.374931e-01
## 13.4   0.1414214   -1.049295e-01    5.809866e-02
## 9      0.1414214   -1.609148e-01   -3.137364e-02
## 42.1   0.1414214    1.190117e-01    3.246256e-02
## attr(,"assign")
## [1] 0 1 1
##
## $qraux
## [1] 1.141421 1.063026 1.088453
##
## $pivot
## [1] 1 2 3
##
## $tol
## [1] 1e-07
##
## $rank
## [1] 3
##
## attr(,"class")
## [1] "qr"
##
## $df.residual
## [1] 47
##
## $xlevels
## named list()
##
## $call
## fit(formula = formula, data = data)
##
## $terms
## dist ~ poly(speed, 2)
## attr(,"variables")
## list(dist, poly(speed, 2))
## attr(,"factors")
##                poly(speed, 2)
## dist                        0
## poly(speed, 2)              1
## attr(,"term.labels")
## [1] "poly(speed, 2)"
## attr(,"order")
## [1] 1
```

```
## attr(,"intercept")
## [1] 1
## attr(,"response")
## [1] 1
## attr(,".Environment")
## <environment: R_GlobalEnv>
## attr(,"predvars")
## list(dist, poly(speed, 2, coefs = list(alpha = c(16.42, 15.4744506260872
## ), norm2 = c(1, 50, 1276.18, 43510.282499334))))
## attr(,"dataClasses")
##           dist poly(speed, 2)
##      "numeric"    "nmatrix.2"
##
## $model
##      dist poly(speed, 2).1 poly(speed, 2).2
## 10     17     -0.151720170     -0.006098425
## 34     76      0.044228389     -0.103231745
## 16     26     -0.095734867     -0.081791466
## 20     26     -0.067742216     -0.105255788
## 45     54      0.184191645      0.115031299
## 17     34     -0.095734867     -0.081791466
## 15     28     -0.123727518     -0.048739012
## 17.1   34     -0.095734867     -0.081791466
## 49    120      0.212184296      0.187448306
## 14     24     -0.123727518     -0.048739012
## 35     84      0.044228389     -0.103231745
## 50     85      0.240176947      0.269453446
## 13     20     -0.123727518     -0.048739012
## 13.1   20     -0.123727518     -0.048739012
## 45.1   54      0.184191645      0.115031299
## 13.2   20     -0.123727518     -0.048739012
## 30     40      0.016235738     -0.118119955
## 42     56      0.100213691     -0.044690926
## 40     48      0.100213691     -0.044690926
## 50.1   85      0.240176947      0.269453446
## 20.1   26     -0.067742216     -0.105255788
## 37     46      0.072221040     -0.078755402
## 49.1  120      0.212184296      0.187448306
## 37.1   46      0.072221040     -0.078755402
## 38     68      0.072221040     -0.078755402
## 49.2  120      0.212184296      0.187448306
## 10.1   17     -0.151720170     -0.006098425
## 41     52      0.100213691     -0.044690926
## 41.1   52      0.100213691     -0.044690926
## 7      18     -0.179712821      0.046130296
## 4      22     -0.263690774      0.260345253
## 28     40     -0.011756914     -0.123420032
## 36     36      0.072221040     -0.078755402
## 43     64      0.100213691     -0.044690926
## 28.1   40     -0.011756914     -0.123420032
## 49.3  120      0.212184296      0.187448306
## 36.1   36      0.072221040     -0.078755402
## 28.2   40     -0.011756914     -0.123420032
## 38.1   68      0.072221040     -0.078755402
## 24     20     -0.039749565     -0.119131976
## 18     34     -0.095734867     -0.081791466
## 36.2   36      0.072221040     -0.078755402
```

```
## 28.3   40     -0.011756914     -0.123420032
## 13.3   20     -0.123727518     -0.048739012
## 1       2     -0.347668728      0.560853407
## 40.1   48      0.100213691     -0.044690926
## 3       4     -0.263690774      0.260345253
## 13.4   20     -0.123727518     -0.048739012
## 9      34     -0.179712821      0.046130296
## 42.1   56      0.100213691     -0.044690926
```

```r
# 3 different ways to extract coef
bSamples[,1]$coefficients
```

```
##     (Intercept) poly(speed, 2)1 poly(speed, 2)2
##         45.9200        172.9296         48.2995
```

```r
bSamples[,1][[1]]
```

```
##     (Intercept) poly(speed, 2)1 poly(speed, 2)2
##         45.9200        172.9296         48.2995
```

```r
bSamples[[1]]
```

```
##     (Intercept) poly(speed, 2)1 poly(speed, 2)2
##         45.9200        172.9296         48.2995
```

h. **(10 marks)** In this question, the `bootstrap()` function implemented using `sapply()` is to be used throughout.

   i. *(2 marks)* Construct a bootstrap sample, called `bSamples`, containing **only** the coefficient estimates from fitting `dist` as a quadratic model of `speed`.

      - use `set.seed(314159)` and `B = 500`.
      - show the coefficients from the **last** bootstrap sample.

      Show your code.

```r
set.seed(314159)
quad_fit <- getFitFn(fit = lm, formula = dist ~ poly(speed,2))
bSamples <- bootstrap(cars, quad_fit, B = 500)
bSamples[1,][[500]]
```

```
##     (Intercept) poly(speed, 2)1 poly(speed, 2)2
##       48.72000       145.30685        10.97672
```

   ii. *(5 marks)* Draw a scatterplot of the coefficient estimates (excluding the intercept estimate) from all 500 bootstrap samples.

      - give appropriate title and axis labels
      - have `x` be the coefficient estimate of the linear term; `y`, the coefficient of the quadratic term.
      - use `pch = 19` and a colour with alpha level 0.4
      - use `{r, out.width = "80%"}` as the header to the R chunk
      - add contours of equal density to the plot.

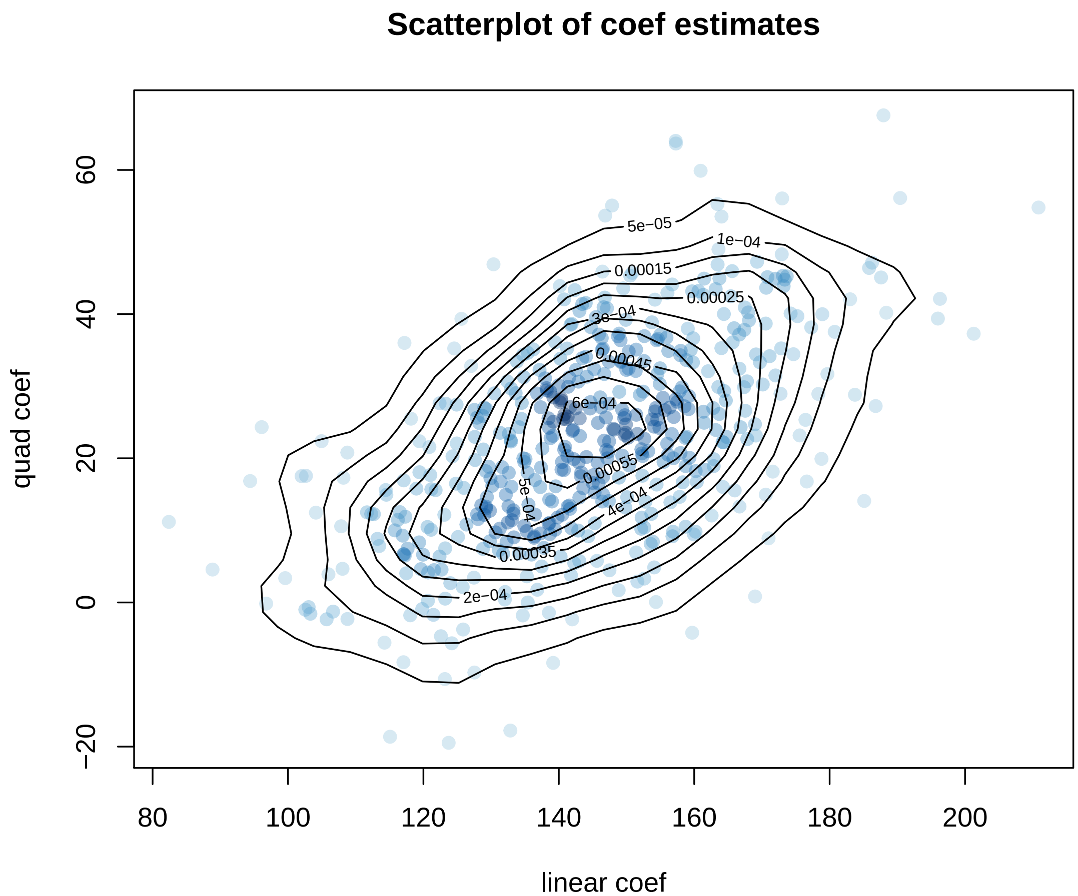
        Note for this

        - you will need the `kde2d()` function from the `MASS` package.
        - use bandwidth function `width.SJ()` for both `x` and `y`.

          (See examples at end of `?kde2d`,)

   Show your code.

```r
library(MASS)
listofCoef <- bSamples[1,]
x <- rep(NA, 500)
y <- rep(NA, 500)
for (i in 1:500){
  x[i] <- listofCoef[[i]][2] #Get linear term coef
  y[i] <- listofCoef[[i]][3] #Get quadratic term coef
}
plot(x,y,type="n", xlab = "linear coef", ylab = "quad coef", main = "Scatterplot of coef estimates")
points(x,y,pch = 19, col = adjustcolor(densCols(x,y),0.4))
f1 <- kde2d(x, y,h = c(width.SJ(x), width.SJ(y)))
contour(f1, add = TRUE)
```



iii. *(3 marks)* Of the two coefficients (linear and quadratic), which does the bootstrap distribution suggest may not be needed? Explain your reasoning.

By looking at the scatterplot and the density, the quadratic coefficients does not suggest neither linear coef nor quad coef is not needed. Because when looking at the higher density parts, both quad and linear coef are not 0. However, if we must pick one coef that is not needed, I would pick the quad coef, because some of the points in the 500 bootstrap samples has 0 as the quad coef, this means that quadratic is not used in that particular sample.