# CS112 Assignment 2

Akmarzhan Abylay

February 2020

## 1 Question 1

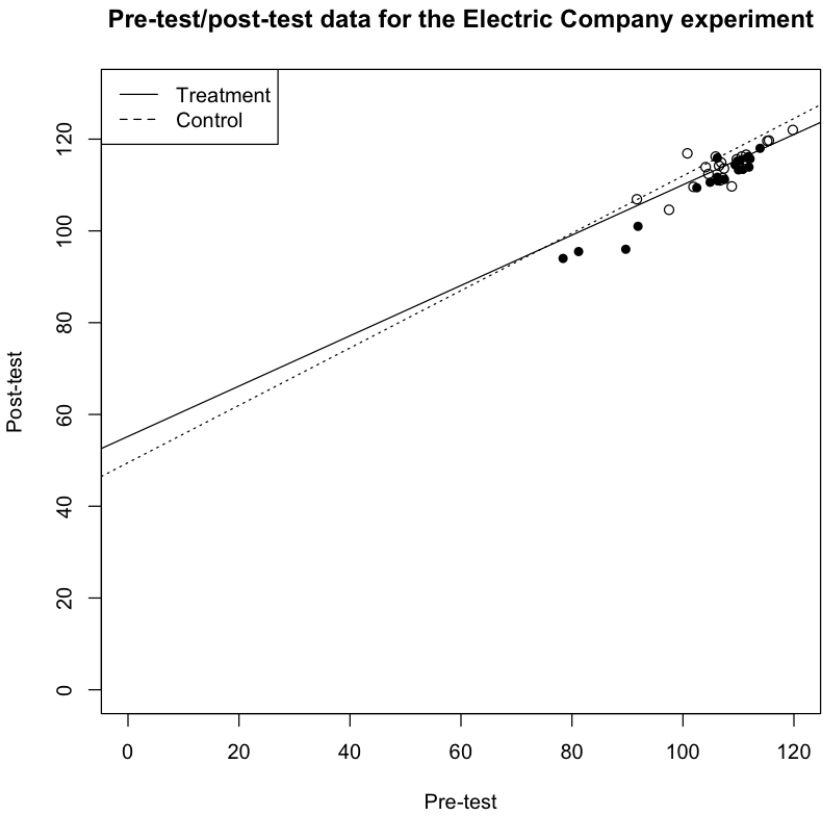
1. Obtain the "Grade 4 Electric Company" data that we used in class during Lesson 3.1:

```
sesame <- read.csv("https://tinyurl.com/wlgl63b")
```

[20]:
```
sesame <- read.csv("https://tinyurl.com/wlgl63b") #loading the data

sesame.treat <- sesame[which(sesame$treatment == 1), ] #dividing the dataset
sesame.ctrl <- sesame[which(sesame$treatment == 0), ]
```

(a) Replicate the "Grade 4" data visualization (the right-panel) in Figure 9.7 of the Gelman reading. Be sure to label everything appropriately.

[18]:
```
#question 1 (a)

lm.sesame.treat <- lm(post.test ~ pre.test, data = sesame.treat) #linear␣
 ↪regression model for treat data

lm.sesame.ctrl <- lm(post.test ~ pre.test, data = sesame.ctrl) #linear␣
 ↪regression model for control data

plot(sesame.treat$pre.test, sesame.treat$post.test, #plotting the treatment data
     main = "Pre-test/post-test data for the Electric Company experiment",
     xlab = "Pre-test",
     ylab = "Post-test", pch=1, xlim=c(0, 120), ylim=c(0, 130))

points(sesame.ctrl$pre.test, sesame.ctrl$post.test, #plotting the control data
     xlab = "Pre-test",
     ylab = "Post-test", pch=16)

abline((lm.sesame.treat), lty=1) #plotting the treat line
abline((lm.sesame.ctrl), lty=3) #plotting the control line

legend("topleft", legend=c("Treatment",
                           "Control"), lty = 1:3)
```




(b) The treatment effects in this "Grade 4" data visualization are positive for all (or nearly all–it's a little hard to tell) X-values shown in the figure. Change the y-value of one data point (in any way you wish) such that the treatment effects are negative for all X values shown in the figure. (Ideally, you would change the y-value just a little, because that would demonstrate the vulnerability of the positive treatment effects to the presence of high-leverage outliers–but ANY change in the given observation is fine.) Create a new figure showing the new (negative) treatment effects AND BE SURE TO MAKE THE POINT YOU MODIFIED A DIFFERENT COLOR FROM ALL THE OTHER POINTS.

Write 1 sentence explaining the data point you changed and the results.

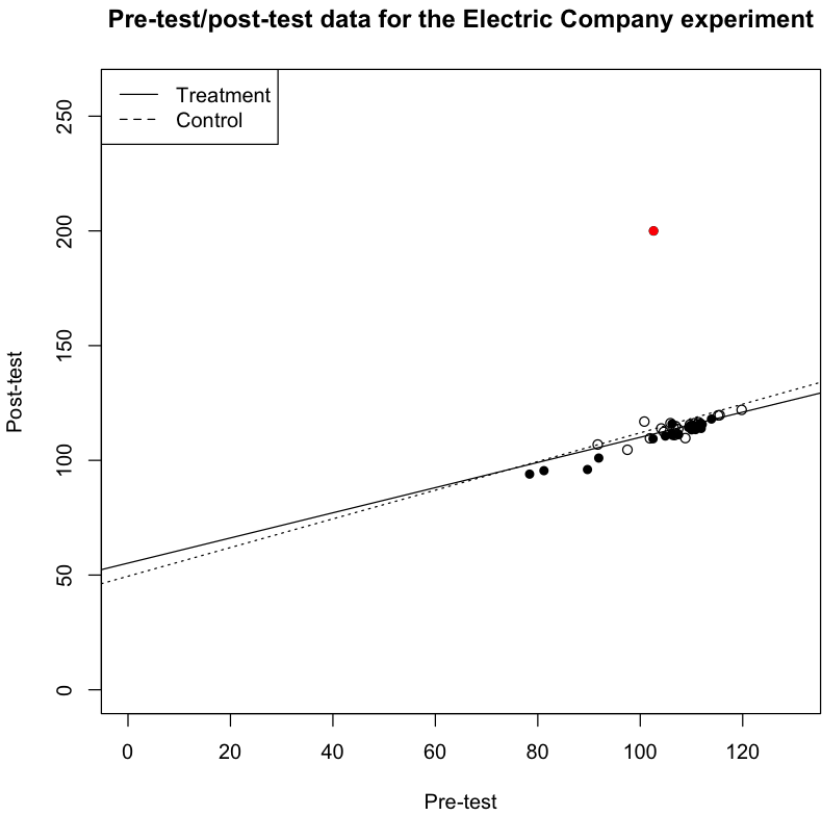
**Answer:**

Below I have changed a point in the control data set where the most observations occur that led to a completely different treatment effect result (now effects are all negative for all X values), which shows the vulnerability of the positive treatment effects to the presence of high-leverage outliers.

[21]:
```
#question 1 (b)

sesame.ctrl[21, 1] <- 200
```

```r
lm.sesame.ctrl <- lm(post.test ~ pre.test, data = sesame.ctrl) #new regression␣
 ↪model
summary(lm.sesame.ctrl)

plot(sesame.treat$pre.test, sesame.treat$post.test, #plotting the treat
     main = "Pre-test/post-test data for the Electric Company experiment",
     xlab = "Pre-test",
     ylab = "Post-test", pch=1, xlim=c(0, 130), ylim=c(0, 260))

points(sesame.ctrl$pre.test, sesame.ctrl$post.test, #plotting the new control␣
 ↪data
       xlab = "Pre-test",
       ylab = "Post-test", pch=16)

points(102.6, 200, xlab = "Pre-test", #outlier
       ylab = "Post-test", col="red", pch=16)

abline((lm.sesame.treat), lty=1) #treat line
abline((lm.sesame.ctrl), lty=3) #new control line

legend("topleft", legend=c("Treatment",
                           "Control"), lty = 1:3)
```

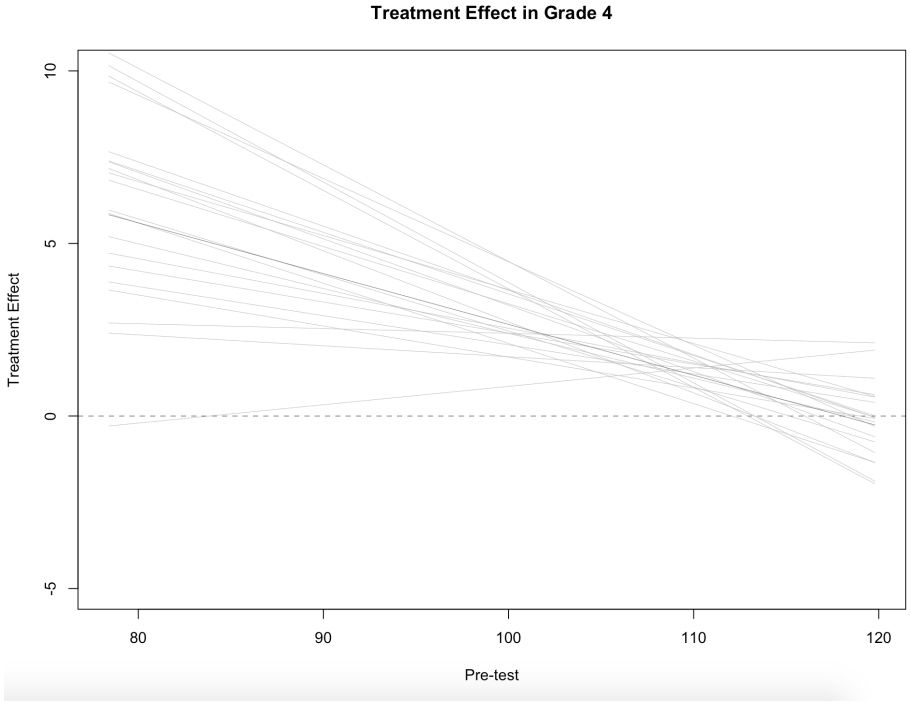
```
Call:
lm(formula = post.test ~ pre.test, data = sesame.ctrl)

Residuals:
   Min     1Q Median     3Q    Max
-9.556 -5.032 -4.169 -3.169 86.381

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  49.4912    46.7385   1.059    0.303
pre.test      0.6250     0.4463   1.400    0.178

Residual standard error: 20.39 on 19 degrees of freedom
Multiple R-squared:  0.09355,Adjusted R-squared:  0.04584
F-statistic: 1.961 on 1 and 19 DF,  p-value: 0.1775
```




(c) Replicate Figure 9.8, again labelling everything appropriately. Do you notice how the uncertainty of the treatment effect estimates is larger at the sides of the figure than it is in the middle? Write a sentence or two discussing and explaining why the uncertainty of the treatment effect is minimized in the region where it appears smallest (i.e., with respect to the X-axis). For extra credit, modify your data visualization to identify (with some color) the 95% interval of the expected value of the treatment effect for all values of X shown in the figure. This requires obtaining simulated expected values for the treatment effect (conditional on pre.test) and then identifying the mid-95% range of those values – for many discrete values of pre.test.

[ ]:
```
#question 1 (c)

library(arm)
set.seed(586)
iterations <- 10000

lm.sesame <- lm(post.test ~ treatment + pre.test + treatment:pre.test, data =␣
 ↪sesame) #linear regression model for all data
lm.sesame.sim <- sim(lm.sesame, n=iterations)

plot(0, 0, xlim=range (sesame$pre.test), ylim=c(-5,10),
```

```
      xlab="Pre-test", ylab="Treatment Effect",
      main="Treatment Effect in Grade 4")
abline (0, 0, lwd=.5, lty=2) #the 0 line

for (i in 1:20){
  curve(lm.sesame.sim@coef[i, 2] + lm.sesame.sim@coef[i, 4]*x, lwd=.5,␣
 ↪col="gray", add=TRUE) #plotting the simulated treatment effect
}
curve(coef(lm.sesame)[2] + coef(lm.sesame)[4]*x, lwd=.5, add=TRUE,␣
 ↪col="dimgrey") #the estimated treatment effect as a function of pre- test␣
 ↪score
```

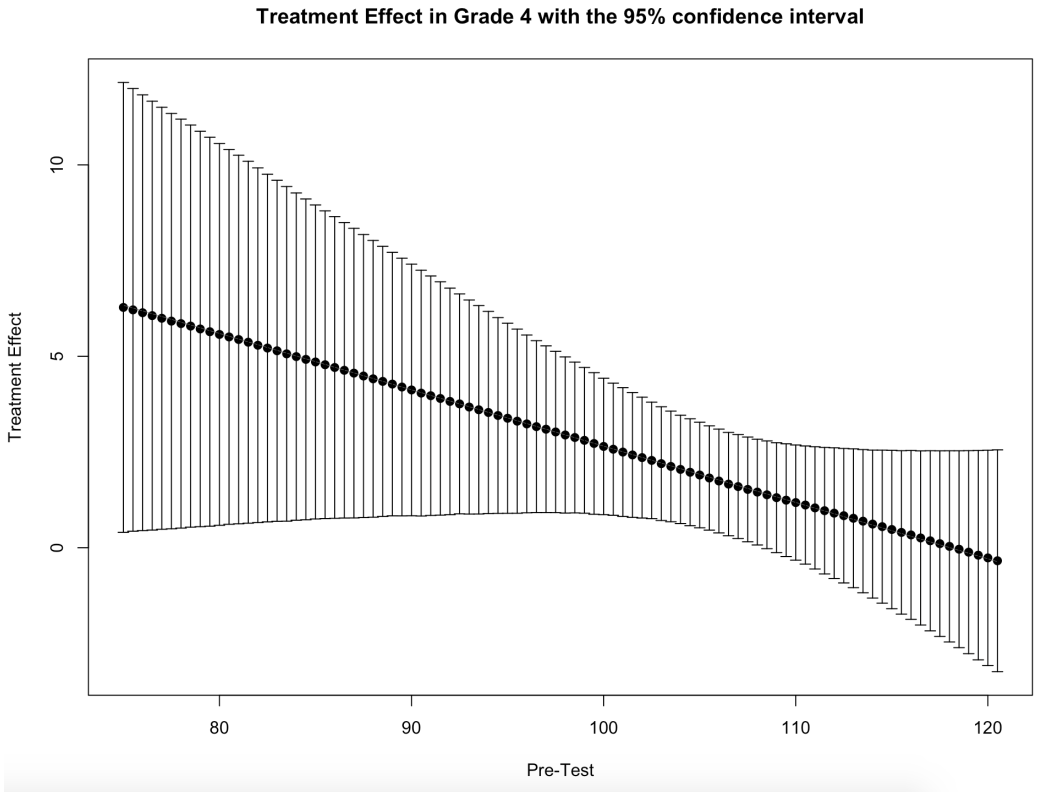



**Answer:**

In the above graph, the uncertainty of the treatment effect is the smallest in the middle and largest on the edges because of the availability of the information for those points. As we could see from the graph in 1a, we have most of the data points gathered around 100 and 110, while only a few points are available on the sides around 80 and 120. If we have more information, we reduce the uncertainty.

[ ]:
```
#question 1 (c) extra
sim.ys_mean.effect <- matrix(NA, nrow = iterations, ncol = 92)

for (index in seq(75, 120.5, by=0.5)){
  for (i in 1:iterations){
```

```
    sim.ys_mean.effect[i, (index-74.5)*2] <- sum(lm.sesame.sim@coef[i, 2],␣
 ↪index*lm.sesame.sim@coef[i, 4])
  }
}
sim.ys_mean.effect

#confidence intervals
CI_mean.effect <- apply(sim.ys_mean.effect, 2, quantile, probs = c(0.025, 0.975))
table.mean.effect <- t(data.frame(CI_mean.effect))
colnames(table.mean.effect) <- c("Lower Bound", "Upper Bound")
table.mean.effect <- data.frame(table.mean.effect)
View(table.mean.effect)

#plotting
plot(seq(75, 120.5, by=0.5), (CI_mean.effect[1, ]+CI_mean.effect[2, ])/2,
     ylim=range(c(CI_mean.effect[1, ], CI_mean.effect[2, ])),
     pch=19, xlab="Pre-Test", ylab="Treatment Effect",
     main="Treatment Effect in Grade 4 with the 95% confidence interval"
)

curve(coef(lm.sesame)[2] + coef(lm.sesame)[4]*x, lwd=.5, add=TRUE, col="dimgrey")
arrows(seq(75, 120.5, by=0.5), CI_mean.effect[1, ], seq(75, 120.5, by=0.5),␣
 ↪CI_mean.effect[2, ], length=0.05, angle=90, code=3)
```



## 2 Question 2

Obtain the "tinting" data set from:

`tinting = read.csv(url("https://tinyurl.com/v4bq99k"))`

Fit a linear model that predicts csoa as a function of `age, sex, target, and I(tint != "no"), and I(as.numeric(tint!= "no")*age)` .

Estimate:

(a) the 95% interval and mean of expected values for `csoa`, for typical TREATED female units, with `target = ''hicon'', ages = 20, 30, 40, 50, 60, 70, and 80` using simulation (i.e., 1000 simulated predictions for every row from 1000 sets of coefficients). In this data set, a "treated unit" is a unit for which `tint != ''no''` is `TRUE`.

(b) b) the 95% interval and mean for the treatment effect, for typical female units, with `target = ''hicon'', ages = 20, 30, 40, 50, 60, 70, and 80` using simulation (i.e., 1000 simulated predictions for every row from 1000 sets of coefficients).

E.g. (with different data): https://gist.github.com/diamonaj/75fef6eb48639c2c36f73c58d54bac2f

**Answer:**

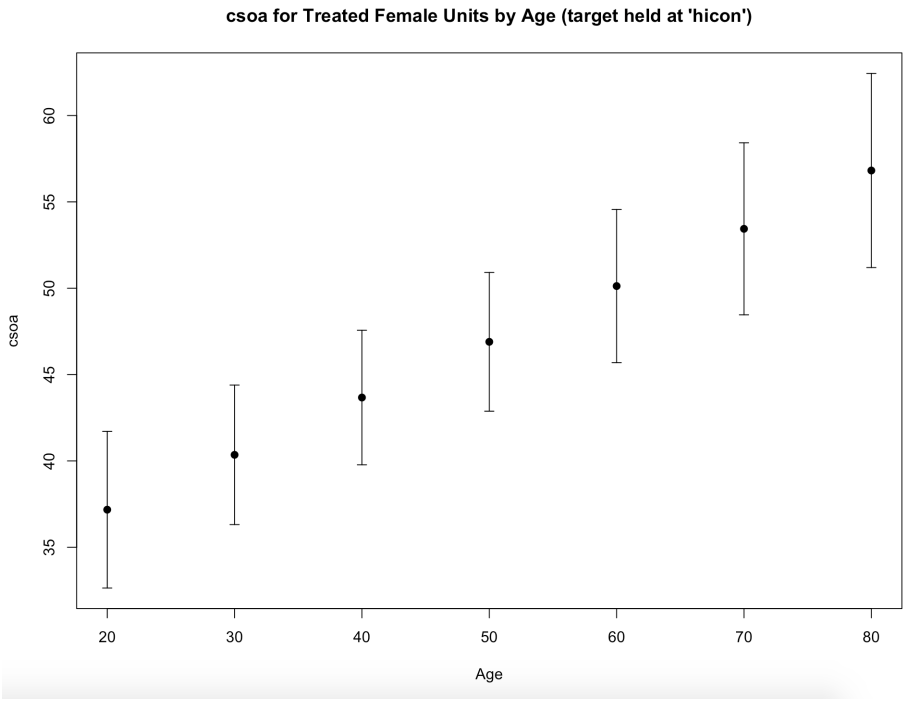
(a) A table with the relevant point estimates: e.g., the bounds of the prediction intervals and means of y for the different ages.

I have tweaked the data a little bit here, changing some of the categorical variables on the dummy numerical variables (binary) so that it is easier to conduct the analysis and for R to be able to work with them when simulating the predictions.

[ ]:
```r
#question 2

tinting <- read.csv(url("https://tinyurl.com/v4bq99k"))

#tint=="no" is now 0 and others are 1, since they are treated
tinting$tint <- ifelse(tinting$tint=="no", 0, 1)
#target="hicon" is 1, while "locon" is 0
tinting$target<- ifelse(tinting$target=="locon", 0, 1)
#sex="f" will be 1 and sex="m" will be 0
tinting$sex<- ifelse(tinting$sex=="f", 1, 0)

lm.tinting <- lm(csoa ~ age + sex + target +I(tint != 0) + I(as.numeric(tint!=␣
 ↪0)*age), data=tinting)
summary(lm.tinting)

#I didn't use the built-in lm function, since it only treats no as no treatment␣
 ↪and lo as treatment, but disregards the hi which makes the population we are␣
 ↪looking at smaller

set.seed(586)
iterations <- 1000
```

```
sim.tinting <- sim(lm.tinting, n.sims = iterations)

# Predict re78 for every unit of age for every set of coefficients holding at␣
 ↪the means
sim.ys_mean.treat <- matrix(NA, nrow = iterations, ncol = length(seq(20, 80,␣
 ↪by=10)))

for (age in seq(20, 80, by=10)){
  Xs <- c(1, age, 1, 1, 1, 1*age)
  for (i in 1:iterations){
    sim.ys_mean.treat[i, (age-10)/10] <- sum(Xs*sim.tinting@coef[i,])
  }
}

#confidence intervals
CI_mean.treat <- apply(sim.ys_mean.treat, 2, quantile, probs = c(0.025, 0.975))
csoa.mean <- (CI_mean.treat[1, ]+CI_mean.treat[2, ])/2
table.mean.treat <- t(data.frame(CI_mean.treat))
colnames(table.mean.treat) <- c("csoa Lower Bound", "csoa Upper Bound")
table.mean.treat <- data.frame(table.mean.treat, csoa.mean)
rownames(table.mean.treat) <- seq(20, 80, by=10)
View(table.mean.treat)

#plotting
plot(seq(20, 80, by=10), (CI_mean.treat[1, ]+CI_mean.treat[2, ])/2,
     ylim=range(c(CI_mean.treat[1, ], CI_mean.treat[2, ])),
     pch=19, xlab="Age", ylab="csoa",
     main="csoa for Treated Female Units by Age (target held at 'hicon')"
)

arrows(seq(20, 80, by=10), CI_mean.treat[1, ], seq(20, 80, by=10), CI_mean.
 ↪treat[2, ], length=0.05, angle=90, code=3)
```

| | csoa.Lower.Bound | csoa.Upper.Bound | csoa.mean |
|---|---|---|---|
| **20** | 32.63994 | 41.71180 | 37.17587 |
| **30** | 36.31304 | 44.39214 | 40.35259 |
| **40** | 39.77392 | 47.56659 | 43.67025 |
| **50** | 42.87907 | 50.91341 | 46.89624 |
| **60** | 45.68857 | 54.56107 | 50.12482 |
| **70** | 48.45891 | 58.41982 | 53.43936 |
| **80** | 51.19638 | 62.43577 | 56.81607 |

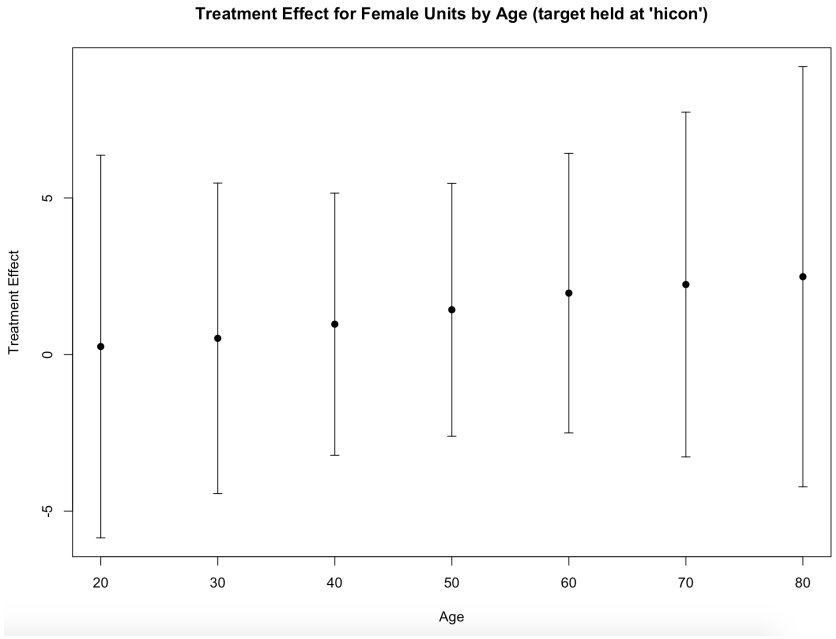



(b) 2 figures for (a) and (b). All I am interested in are the prediction intervals and means for each age. The figures should show how the intervals change over time (i.e., over the range of ages in the data set). Be sure to label your plot's features (axis, title, etc.).

[ ]:
```
#question 2 (b)

set.seed(586)
iterations <- 1000
sim.tinting <- sim(lm.tinting, n.sims = iterations)

# Predict re78 for every unit of age for every set of coefficients holding at␣
 ↪the means
sim.ys_mean.te <- matrix(NA, nrow = iterations, ncol = length(seq(20, 80,␣
 ↪by=10)))

for (age in seq(20, 80, by=10)){
  X.treat <- c(1, age, 1, 1, 1, age)
  X.ctrl <- c(1, age, 1, 1, 0, 0*age)
  for (i in 1:iterations){
    sim.ys_mean.treat[i, (age-10)/10] <- sum(X.treat*sim.tinting@coef[i,])-sum(X.
 ↪ctrl*sim.tinting@coef[i,])
  }
}

#confidence intervals
CI_mean.te <- apply(sim.ys_mean.treat, 2, quantile, probs = c(0.025, 0.975))
```

```r
table.mean.te <- t(data.frame(CI_mean.te))
treatment.mean <- (CI_mean.te[1, ]+CI_mean.te[2, ])/2
colnames(table.mean.te) <- c("Treatment Effect Lower Bound", "Treatment Effect␣
 ↪Upper Bound")
table.mean.te <- data.frame(table.mean.te, treatment.mean)
rownames(table.mean.te) <- seq(20, 80, by=10)
View(table.mean.te)

#plotting
plot(seq(20, 80, by=10), (CI_mean.te[1, ]+CI_mean.te[2, ])/2,
     ylim=range(c(CI_mean.te[1, ], CI_mean.te[2, ])),
     pch=19, xlab="Age", ylab="Treatment Effect",
     main="Treatment Effect for Female Units by Age (target held at 'hicon')"
)

arrows(seq(20, 80, by=10), CI_mean.te[1, ], seq(20, 80, by=10), CI_mean.te[2, ],␣
 ↪length=0.05, angle=90, code=3)
```

| | Treatment.Effect.Lower.Bound | Treatment.Effect.Upper.Bound | treatment.mean |
|---|---|---|---|
| **20** | -5.854292 | 6.364298 | 0.2550030 |
| **30** | -4.441076 | 5.475094 | 0.5170088 |
| **40** | -3.217260 | 5.154751 | 0.9687457 |
| **50** | -2.607742 | 5.466559 | 1.4294083 |
| **60** | -2.502499 | 6.424977 | 1.9612389 |
| **70** | -3.268835 | 7.740523 | 2.2358439 |
| **80** | -4.223831 | 9.194141 | 2.4851546 |



## 3 Question 3

3. Write your own bootstrap function (5 lines max) that takes Ys and predicted Ys as inputs, and outputs $R^2$. Copy/paste an example using the afterschool data (from #3 above) that shows it working - (i.e., show that the summary(lm()) command outputs a very similar $R^2$ as your own function does..

[ ]:
```r
#question 3

library(Matching)
data(lalonde)

fit <- lm(re78 ~ age, data = lalonde)
iterations <- 100000
boot_storage.r_sqr <- rep(NA, iterations)

# bootstrap function
rsquared <- function(y.true, y.pred) {
  for (i in 1:iterations){
    sample.r_sqr <- sample(1:length(y.true), length(y.true), replace = TRUE)
    rss <- sum((y.true[sample.r_sqr] - y.pred[sample.r_sqr])**2)
    tss <- sum((y.true[sample.r_sqr] - mean(y.true[sample.r_sqr]))**2)
    boot_storage.r_sqr[i] <- 1 - rss/tss #based on the r-squared formula
  }
  return(c(mean(boot_storage.r_sqr), quantile(boot_storage.r_sqr, c(0.025, 0.
↪975))))
}

rsquared(lalonde$re78, fit$fitted.values)
summary(fit)$r.sq
```

```
> rsquared(lalonde$re78, fit$fitted.values)
                      2.5%        97.5%
 0.001125828 -0.011411737  0.010785053
> summary(fit)$r.sq
[1] 0.003267913
```

## 4 Question 4

4. Bootstrap the 95% confidence interval of the treatment effect using the mazedata1.dta data (define the binary treatment condition as "Caste Revealed" vs. not "Caste Revealed, and define outcomes as "round1". No covariates (independent variables) are needed. Use 10000 bootstrap samples.

Don't use a 'canned' bootstrap function – please code the bootstrap routine manually.

Then, obtain the a conventional confidence interval for the treatment effect using the "confint(lm())" function.

Your answer to this question should consist of the following:

(a) A table with the relevant results (bounds on the 2 confidence intervals–one obtained via bootstrapping, the other obtained via confint).

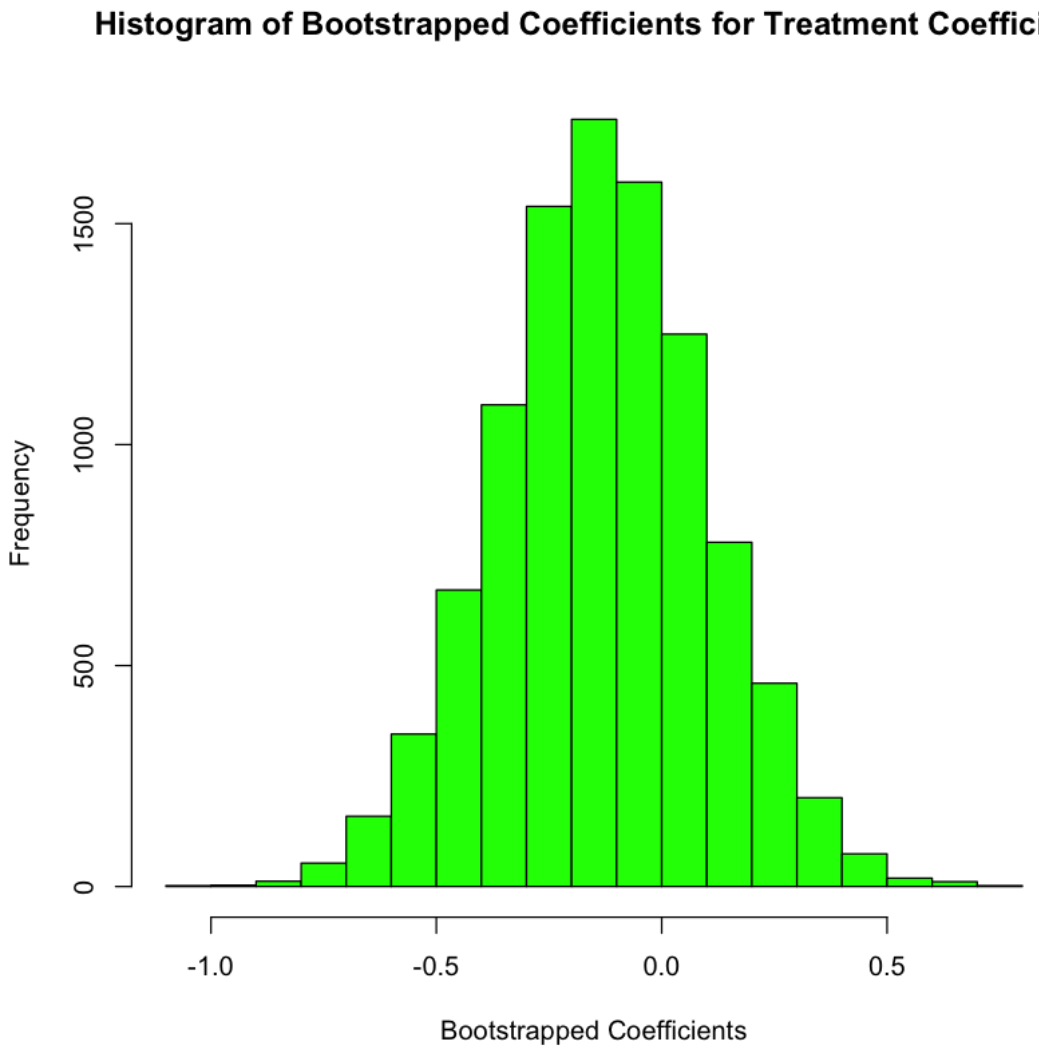
[15]:
```
#question 4a

library(foreign)
data <- read.dta('Downloads/mazedata1.dta')

data$treatment<- ifelse(data$treatment=="Caste Revealed", 1, 0)

iterations <- 10000
boot_storage <- rep(NA, iterations)
for (i in 1:iterations) {
  lm.boot = lm(round1 ~ treatment, data = data[sample(1:nrow(data), nrow(data),␣
 ↪replace = TRUE),])
  boot_storage[i] <- lm.boot$coefficients[2]
}

#bootstrapped confidence interval
quantile(boot_storage, c(0.025, 0.975))

#analytical confidence interval
lm.analytical <- lm(round1 ~ treatment, data = data)
confint(lm.analytical)[2,]

table <- data.frame(quantile(boot_storage, c(0.025, 0.975)), confint(lm.
 ↪analytical)[2,])
rownames(table) <- c("Lower Bound", "Upper Bound")
colnames(table) <- c("Bootstrapped", "Analytical")
View(table)
```

| | Bootstrapped | Analytical |
|---|---|---|
| **Lower Bound** | -0.5923809 | -0.5842684 |
| **Upper Bound** | 0.3220887 | 0.3194536 |

(b) A histogram (properly labeled) showing your bootstrap-sample results. How you do this one is up to you.

[16]:
```
#question 4b
hist(boot_storage,n=20,
```

```
    main="Histogram of Bootstrapped Coefficients for Treatment Coefficient",
    xlab="Bootstrapped Coefficients",
    col="green")
```




## 5 Question 5

5. Obtain the nsw.dta dataset from:

```
foo <- read.csv(url("https://tinyurl.com/yx8tqf3k"))
```

Randomly remove 2000 observations, and set it aside as a test set (or validation set). Do so using the following code:

```
set.seed(12345)
```

`test_set_rows <- sample(1:length(foo$age), 2000, replace = FALSE)`

Use 10-fold cross validation and LOOCV to estimate the test-set error (MSE) for your 2 models. Then compare the cross validation test-set error estimates to the test-set error you obtain when you actually run your model on your 2000-row test set.

Provide a table that summarizes the 6 different sets of numbers (2 10-fold CV estimates, 2 LOOCV estimates, and 2 test set error rates). Explain your results in no more than a paragraph. Contrast the time it takes to run 10-fold CV versus LOOCV and explain why one takes much longer.

[ ]:
```r
#question 5
foo <- read.csv(url("https://tinyurl.com/yx8tqf3k"))

set.seed(12345)
library(boot)
test_set_rows <- sample(1:length(foo$age), 2000, replace = FALSE)
foo.test <- foo[test_set_rows,]
foo.train <- foo[-test_set_rows,]

# models
glm.fit1 <- glm(treat ~ age, data=foo.train, family=binomial)
glm.fit2 <- glm(treat ~ age+ age^2 + education + age:education + married + u74 +␣
 ↪married:u74, data=foo.train, family=binomial)

# LOOCV and 10-fold CV
cv.err1.k <- cv.glm(foo.train, glm.fit1, K=10)
cv.err1.k$delta
cv.err1.loocv <- cv.glm(foo.train, glm.fit1)
cv.err1.loocv$delta

cv.err2.k <- cv.glm(foo.train, glm.fit2, K=10)
cv.err2.k$delta
cv.err2.loocv <- cv.glm(foo.train, glm.fit2)
cv.err2.loocv$delta

#MSE of the test set
ts1 <- mean((foo.test$treat - predict(glm.fit1, foo.test, type="response"))^2)
ts2 <- mean((foo.test$treat - predict(glm.fit2, foo.test, type="response"))^2)

cv.table <- data.frame(c(cv.err1.k$delta[1], cv.err2.k$delta[1]), c(cv.err1.
 ↪loocv$delta[1], cv.err2.loocv$delta[1]), c(ts1, ts2))
colnames(cv.table) <- c("LOOCV", "10-FOLD", "Test Set Error")
rownames(cv.table) <- c("Model 1", "Model 2")
View(cv.table)
```

| | LOOCV | 10-FOLD | Test Set Error |
|---|---|---|---|
| **Model 1** | 0.01124668 | 0.01124643 | 0.01131240 |
| **Model 2** | 0.01069548 | 0.01069843 | 0.01073288 |

**Answer:**

As we could see from the table above, both errors are very similar and close to the test set error, but the LOOCV error is a bit closer for model 1 (see relative error table below). However, for the second model, the error of the 10-fold is closer to the actual test set error.

Error rates are calculated by formula $\eta = \frac{|measured - actual|}{actual}$ and are as follows:

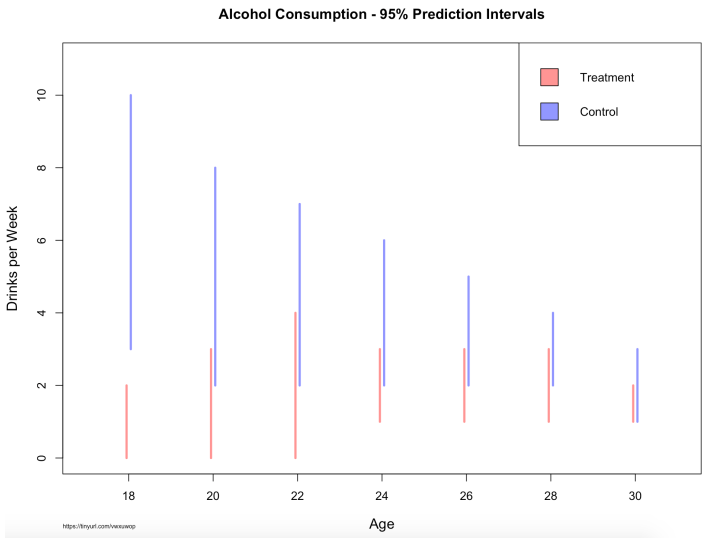
| Model | Relative Error (LOOCV) | Relative Error (10-fold) |
|---|---|---|
| #1 | 0.00581 | 0.00583 |
| #2 | 0.00348 | 0.00321 |

In both cases, there is only a small relative error from the actual test set error, so there wasn't a significant difference in the results from the LOOCV and 10-fold. However, taking time into account, we would say 10-fold is much better than the LOOCV, since while 10-fold runs in second, the LOOCV takes more than 30 minutes, which is not a very good time investment for such a small difference and possibly worst relative error.

Here, 10-fold takes much less time than LOOCV, since in 10-fold, we are only considering 10 clusters of data and cross-validating through them, each time taking a new cluster as the validation set, while in LOOCV, we need to go through 14177 such "clusters", since we are leaving one out as a validation set and our k=n, which in this case is 14177. Logically, it would take much more time to iterate that many times, which is why LOOCV takes more than half an hour to run.

# 6 Question 6

6. Write the executive summary for a decision brief about the impact of an alcohol-awareness program targeted at individuals age 18-30, intended to reduce binge drinking. The program was tested via RCT, and the results are summarized by the figure that you get if you run this code: https://tinyurl.com/sj3mp76. You executive summary should be between about 4 and 10 sentences long, it should briefly describe the the purpose of the study, the methodology, and the policy implications/prescription. (Feel free to imaginatively but realistically embellish/fill-in-the-blanks with respect to any of the above, since I am not giving you backstory here).




Here we are considering the impact of an alcohol-awareness program targeted at individuals aged 18-30, intended to reduce binge drinking. The purpose of the study is to see whether the alcohol-awareness program helps reduce binge drinking.

**Methodology:**

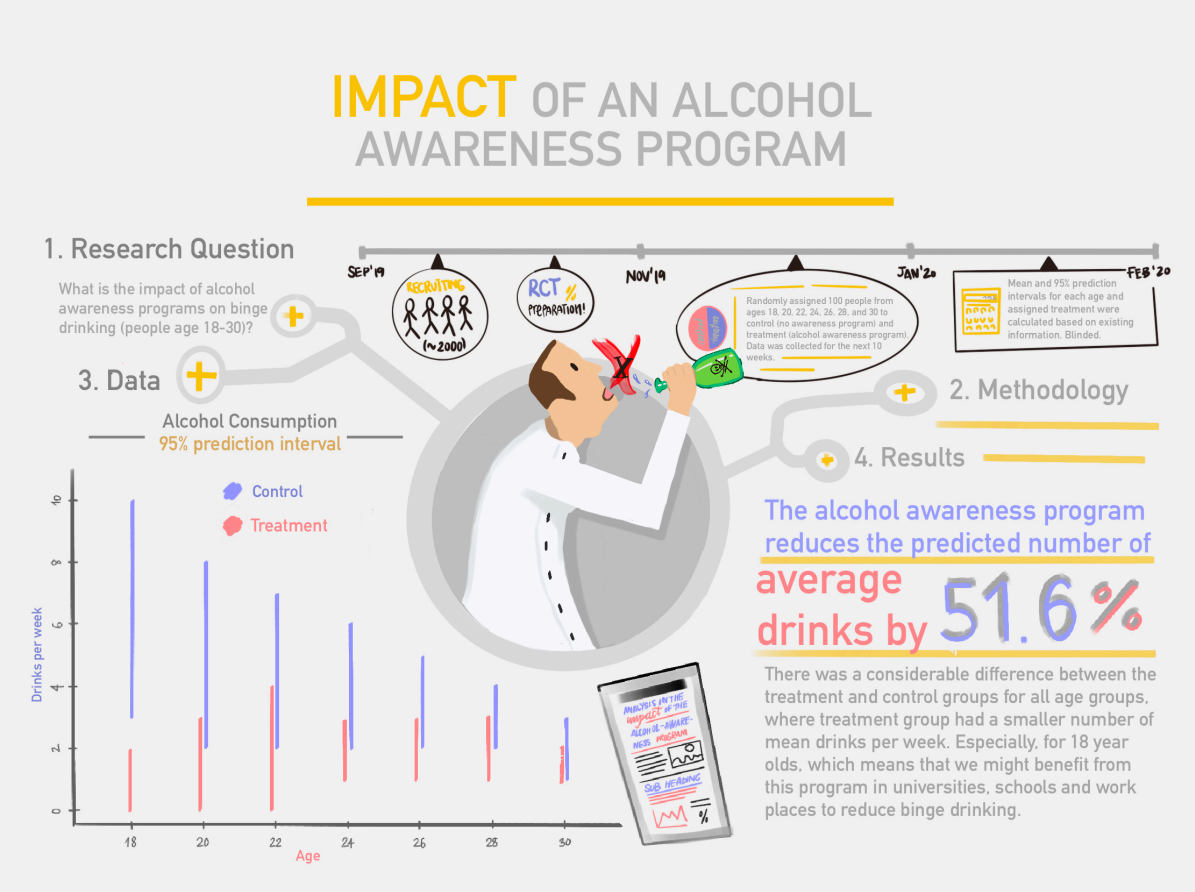
A randomized controlled trial was conducted. We took a sample of 100 people (i.e., volunteers) with different drinking habits, from ages 18-30 (only the even ages are considered - 18, 20, 22, 24, 26, 28, 30) and randomly assigned treatment (i.e., going through an alcohol-awareness program) to half. In contrast, the others were assigned control (i.e., no program). Right after the alcohol-awareness program was facilitated, the data about people's drinking was collected for the next 10 weeks and averaged. The mean of all participants was identified, and then the 95% prediction intervals were plotted for each group based on the current information. The study was blinded, meaning that the people who have analyzed the data didn't know which subjects were assigned which treatment to prevent overestimation of the treatment effect. The possible biases (e.g., selection bias, examiner bias, etc.) were hopefully reduced by random assignment and blinding.

**Policy implications:**

We saw that there was a difference between the treatment and control group predictions for all age groups, where treatment had a considerably smaller number of predicted drinks per week (especially for 18-year-olds). Although we cannot guarantee or predict 100% causality, and as a result of lack of information and a possibility of the lack of external validity, these results might be irrelevant. However, it might be beneficial to conduct alcohol-awareness programs in schools, universities, and workplaces to reduce binge drinking. Also, from data we have, we see that the program reduces the predicted alcohol consumption by 51.6%, on average (i.e., dividing the difference in the means for each age by the control group average drink count for each age and averaging them).

**Bonus**:

I also drew some simple info graphics to reflect the findings of the study. You can see a better quality version here.




## 7 Question 7

7. (Optional) Figure out how to use the rgenoud genetic algorithm (you'll have to install the rgenoud library) to run a least-squares regression. You'll have to write a function that calculates the sum of the squared residuals, and then you will feed that function to rgenoud. Demonstrate that your genetic algorithm obtains the same (or very nearly the same) coefficients as using the lm function.

```
library(rgenoud)

data_outlier = read.csv(url("https://tinyurl.com/vfsoe4q"))
```

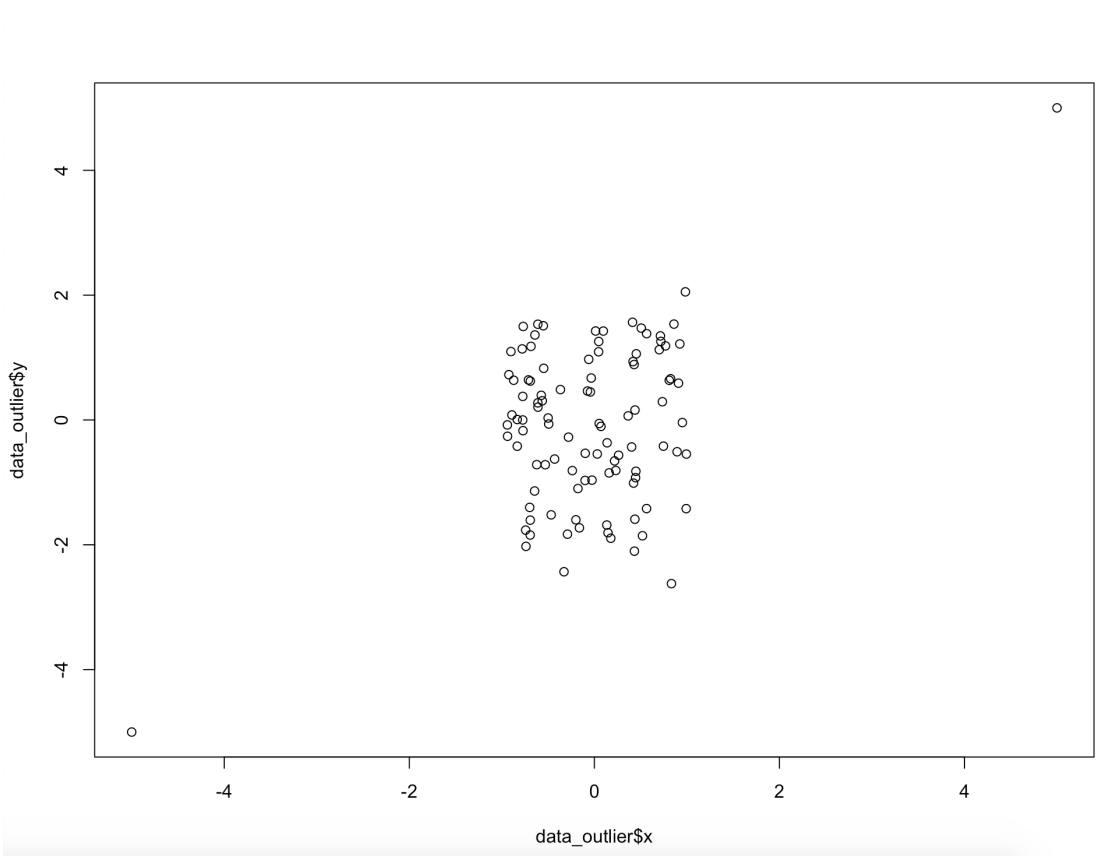
Then, once you've established that your code is working correctly, write another function to run a least-absolute-deviations regression (in which the genetic algorithm identifies the linear function that produces the smallest sum of ABSOLUTE residuals). NOTE: You must add an argument "gradient.check = FALSE" to the genetic algorithm to ensure that it runs properly (You are not expected to know or explain why).

Comment on the difference between the output (slope and intercept) of the two functions. Is one function more sensitive to the existing outlier? Why?

[ ]:
```
#question 7
```

```
library("Matching")
library("rgenoud")
data(lalonde)

data_outlier <- read.csv(url("https://tinyurl.com/vfsoe4q")) #loading data

#least squares function
ls_regr <- function(coefs) {
  rss <- sum((data_outlier$y - (coefs[1] * 1 + coefs[2]*data_outlier$x))^2)
  return(rss)
}

gen.rss <- genoud(fn = ls_regr, nvars = 2,␣
 ↪Domains=matrix(c(-500,500,-500,500),2,2,byrow = TRUE),
                boundary.enforcement = 2, pop.size = 100,
                max.generations = 100, wait.generations = 5)
gen.rss$par

lm.outlier <- lm(y ~ x, data=data_outlier)
lm.outlier$coef

#smallest sum of absolute residuals function
sae_regr <- function(coefs) {
  sae <- sum(abs(data_outlier$y - (coefs[1] * 1 + coefs[2]*data_outlier$x)))
  return(sae)
}

#plotting
gen.sae <- genoud(fn = sae_regr, nvars = 2,␣
 ↪Domains=matrix(c(-500,500,-500,500),2,2,byrow = TRUE),
                gradient.check = FALSE, boundary.enforcement = 2, pop.size = 100,
                max.generations = 100, wait.generations = 5)
gen.sae$par
gen.rss$par
lm.outlier$coef

plot(data_outlier$x, data_outlier$y)
```

```
> gen.sae$par
[1] 0.1560311 0.4476885
> gen.rss$par
[1] -0.06867333  0.63542130
> lm.outlier$coef
(Intercept)           x
-0.06867333  0.63542130
```




**Answer:**

If we look at the results of the linear regression without the outliers, we could see that the actual trend in the leftover 99 points is negative rather than positive, which was shown by the absolute residuals above. The R built-in function gives the same result as the RSS function since they use the same method. This might suggest that the sum of the line of the sum of the absolute residuals is more sensitive to the outliers. However, RSS is sensitive to outliers too, and in some cases, it is even less robust when we don't have enough data except the outliers in the dataset. In this exact dataset, RSS turned out to give better results.

```
> gen.sae$par
[1] -0.01903886 -0.02401955
> gen.rss$par
[1] -0.09177905  0.10704008
> lm.outlier$coef
(Intercept)           x
-0.09177905  0.10704008
```

This makes sense if we think about the fact that RSS puts more emphasis on the miscalculated quantities (the difference between the predicted and the actual outcomes) since it is squaring

the difference, which makes it more sensitive to the points in the whole dataset. But intuitively, shouldn't it make it more sensitive to the outliers? If the outliers are way out of the zone of the most data clusters, then yes, it makes a huge difference. This happens because, in terms of the final error, the line will try to minimize that and end up being completely different than the actual value, as we saw from several examples in class. However, in this case, we have enough points in the main cluster (where most of the points lie), and the outliers are not too far away from the cluster so that the RSS line is not too tweaked. Although it is sensitive to the outliers, it also has 99 other points to care about, which makes it more robust since the residuals for them are squared too.

At the same time, for the sum of absolute residuals, the summed difference between the predicted and actual value for the main cluster is not enough to keep the line robust since the outliers outweigh it anyways, which is why we have the slope predicted by the least sum of absolute residuals as positive instead of negative.

I also made my own genetic algorithm in Python, which is shown below. I tried to use both the least squared residuals sum and the least absolute sum of residuals.

[ ]:
```
import numpy as np
import pandas as pd
import random
#imports the random library import numpy as np
#imports a numpy library

#loads the training data
my_data_file = "https://tinyurl.com/vfsoe4q" #assigns the file name to a␣
 ↪variable "my_data_file"
train = pd.read_csv(my_data_file)
```

[ ]:
```
from random import random, sample, choice
from math import floor
from numpy import array, dot, mean
from numpy.linalg import pinv

def generate_data(): #inputting the data points from soil data set
    return x, y2

def crechromosome(ind_size): #creating chromosomes for the population
    return [np.random.choice(np.linspace(0,0.2,num = 15000)) for i in␣
 ↪range((ind_size))]


def crepop(ind_size, population_size): #creating a population of solutions based␣
 ↪on created chromosomes
    return [crechromosome(ind_size) for i in range(population_size)]

def fitness(individual, inputs): #assessing individual's fit based on the found␣
 ↪SSR
```

```
    predicted_outputs = dot(array(inputs), array(individual))
    output_mean = mean(outputs)
    SSR = array([(i - j) ** 2 for i, j in zip(outputs, predicted_outputs)]
        ).sum()
    average_error = (SSR / len(outputs))
    return {'The average error of the regression line': average_error, 'coeff':
 ↪individual}


def evaluate(population): #evaluating the fitness values and returning the best␣
 ↪representatives list (survivors)
    fitness_list = [fitness(individual, inputs)
                    for individual in (population)]
    error_list = sorted(fitness_list, key=lambda i: i['The average error of the␣
 ↪regression line'])
    best_individuals = error_list[: selection_size]
    best_individuals_stash.append(best_individuals[0]['coeff'])
    return best_individuals


def crossover(parent_1, parent_2): #applying crossover and returning the␣
 ↪chromosome (child) that has been changed
    child = {}
    sol = [i for i in range(0, ind_size)]
    sol_1 = sample(sol, floor(0.5*(ind_size)))
    sol_2 = [i for i in sol if i not in sol_1]
    chromosome_1 = [[i, parent_1['coeff'][i]] for i in sol_1]
    chromosome_2 = [[i, parent_2['coeff'][i]] for i in sol_2]
    child.update({key: value for (key, value) in chromosome_1})
    child.update({key: value for (key, value) in chromosome_2})
    return [child[i] for i in sol]


def mutate(individual): #applying mutation and returning the mutated chromosome
    sol = [i for i in range(0, ind_size)]
    no_of_genes_mutated = floor(probgenmut*ind_size)
    sol_to_mutate = sample(sol, no_of_genes_mutated)
    for solus in sol_to_mutate:
        gene_transform = choice([-1, 1])
        change = gene_transform*random()
        individual[solus] = individual[solus] + change
    return individual


def get_new_gen(selected_individuals): #applying both crossover and mutation and␣
 ↪returning the changed offspring
    parent_pairs = [sample(selected_individuals, 2)
```

```
                    for i in range(population_size)]
    offspring = [crossover(pair[0], pair[1]) for pair in parent_pairs]
    offspring_indices = [i for i in range(population_size)]
    offspring_to_mutate = sample(offspring_indices,␣
 ↪floor(probindmut*population_size))
    mutated_offspring = [[i, mutate(offspring[i])]
                         for i in offspring_to_mutate]
    for child in mutated_offspring:
        offspring[child[0]] = child[1]
    return offspring

def check_term(best_individual): #checking the termination condition and␣
 ↪stopping the generations when reached 50
    if generation_count == max_generations:
        return True
    else:
        return False

y2=np.array(list(train["y"].values))
x0=np.array(list(train['x'].values)) #turns the dataframe into a list for easier␣
 ↪analysis

x=np.zeros((101, 2))

for i in range(101):
    x[i][0]=x0[i]
    x[i][1]=1



#main program
inputs, outputs = generate_data()
ind_size = len(inputs[0])
population_size = 1000
selection_size = floor(0.1*population_size)
max_generations = 100
probindmut = 0.1
probgenmut = 0.15
best_individuals_stash = [crechromosome(ind_size)]
initial_population = crepop(ind_size, 1000)
current_population = initial_population
termination = False
generation_count = 0

#finding and printing the solution
while termination is False: #running until the generation count hits 50
```

```
    current_best_individual = fitness(best_individuals_stash[-1], inputs)␣
↪#finding the current best individual
    best_individuals = evaluate(current_population) #finding the best individuals
    current_population = get_new_gen(best_individuals) #creating a new␣
↪population from best representatives
    termination = check_term(current_best_individual) #checking terms for␣
↪termination
    generation_count += 1 #updating the generations count
else:
    print(fitness(best_individuals_stash[-1], inputs))
```

```
{'The average error of the regression line': 1.5581116992063 87, 'coeff': [0.19943996266417763, 0.000240016001066737
8]}
```

The below result is for the absolute residuals.

```
{'The average error of the regression line': 1.0126895984772522, 'coeff': [0.19906660444029603, 0.03228215214347623]}
```

The results are not much different for the least sum of absolute residuals. However, it is very different for the least-squares sum of the residuals, for some reason. And the error for the RSS is higher than for the second function. This might be because of the difference in the execution of the function, but for me, there was no difference in the way the code was written. However, there might be differences in how the genetic algorithm works for R and in my Python code.

## 8 Code

You can find the code here.