exam_25

2025-06-02

Task 1

a)

```r
# X is a n x p matrix with data
# y is an n x 1 vector with data

f <- function(b1, X, y, la) {
  fval <- sum((y - X %*% b1)^2) + la * sum(b1^2) # This part calculates the
  # sum of the y vector minus the X (n x p) matrix times (p x 1) vector
  # b1 (the beta coeffcient(coeff)) of different predictors) and
  # the last part is a "la" a lambda that is multiplied with the coeff^
  # la acts as a penalty, since we in the g fucntion want
  # to minimize the fval
  return(fval)
}

g <- function(X, y, la) {
  q <- ncol(X) # The number of predictors
  b0 <- mean(y) # the average of y
  yd <- y - b0 # y after adjusting for the mean(y)
  Xd <- X # storing the variable X in another variable
  for (k in 1:q) Xd[, k] <- X[, k] - mean(X[, k])
  # demeaning the X and storing as Xd
  b1start <- rep(0, q) # Q is the number of predictors
  opt <- nlminb(b1start, f, X = Xd, y = yd, la = la)
  # nlmnib is a optimization function using the function f,
  # the adjusted Xd, yd and la.
  # The objective is to minimize the function.
  b1 <- opt$par # Returns the best set of parameters, or beta coeff.
  return(b1)
}
```

This is a very close version of a ridge regression. Normally you scale the variables for ridge regression, I see that is not done here, and I am not surprised if the optimization method "PORT" is somewhat different than in the normal ridge regression.

```r
library(tidyverse)
```

b)

```
Xy <- read_csv2("data_task1.csv")
```

```
## i Using "','" as decimal and "'.'" as grouping mark. Use `read_delim()` for more control.

## Rows: 100 Columns: 21
## -- Column specification --------------------------------------------------------
## Delimiter: ";"
## dbl (21): y, X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X1...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
X <- as.matrix(Xy[, -1])

y <- as.matrix(Xy[, 1])
```

```
model_lm <- lm(y ~ X)

model_g100 <- g(X, y, 100)

model_g0 <- g(X, y, 0)


coef(model_lm)
```

```
## (Intercept)         XX1         XX2         XX3         XX4         XX5
## 2251.656480   26.198055  -17.161224   21.793449   61.559375    8.023438
##         XX6         XX7         XX8         XX9        XX10        XX11
##  -18.216741   24.787363   51.743651   35.769806  -26.239930  -32.985179
##        XX12        XX13        XX14        XX15        XX16        XX17
##  -12.264092    3.535228  -43.238530   26.127038   32.451475  -10.730619
##        XX18        XX19        XX20
##  -41.533946    3.477019  -24.029345
```

```
model_g0
```

```
##  [1]  26.198025 -17.161156  21.793308  61.559324   8.023529 -18.216689
##  [7]  24.787355  51.743616  35.769826 -26.239897 -32.985128 -12.264121
## [13]   3.535272 -43.238444  26.126900  32.451465 -10.730496 -41.533931
## [19]   3.477005 -24.029347
```

```
model_g100
```

```
##  [1]  25.642829 -14.357235  19.185465  57.913511   9.444029 -16.502958
##  [7]  23.886187  48.674568  36.332764 -23.733590 -31.215757 -11.133745
## [13]   5.610030 -39.718169  21.714096  31.717906  -7.132566 -38.498087
## [19]   2.253281 -22.191593
```

```r
model_g0 - model_g100
```

```
##  [1]  0.5551964 -2.8039212  2.6078431  3.6458128 -1.4205000 -1.7137312
##  [7]  0.9011677  3.0690479 -0.5629374 -2.5063075 -1.7693713 -1.1303762
## [13] -2.0747578 -3.5202747  4.4128032  0.7335597 -3.5979300 -3.0358434
## [19]  1.2237243 -1.8377535
```

The beta coeffcient(coeff) of the lm model is quite similar to the beta coeff from the g-function when the la = 0. There are some differences but these are so small that we can ignore this.

The beta coeffs from g-function with la = 100, however is somewhat different since the shrinkage of the la is applied.

The reason for the difference is "la". It is the shrinkage penalty that reduces the beat coeffs.

c)

```r
loo <- function(la, X, y) {
  n <- nrow(X) # Number of obs in each predictor from X
  q <- ncol(X) # Number of predictors in X
  b0 <- mean(y) # The average of Y
  yd <- y - b0 # y minus the average of y
  Xd <- X # Xd variable a replica of X
  for (k in 1:q) Xd[, k] <- X[, k] - mean(X[, k]) # Adjust the X predictors for
  # the average of each predictor
    ydpred <- matrix(0, n, 1) # A empty vector with length(n)
  for (i in 1:n)
  {
    b1 <- g(X = X[-i, ], y = y[-i], la) # Finds the coeff with the g-function,
    # and excludes the test data
    ydpred[i] <- Xd[i, ] %*% b1 # Multiplies with the Xd to predict the y,
    # with the new data of the test data
  }
  testMSE <- mean((yd - ydpred)^2) # Calculates the MSE as a metric of how good
  # the prediction are on average across all predictions.
  return(testMSE)
}
```

The function above is a leave one out cross validation for the g function.

What about the Xd?

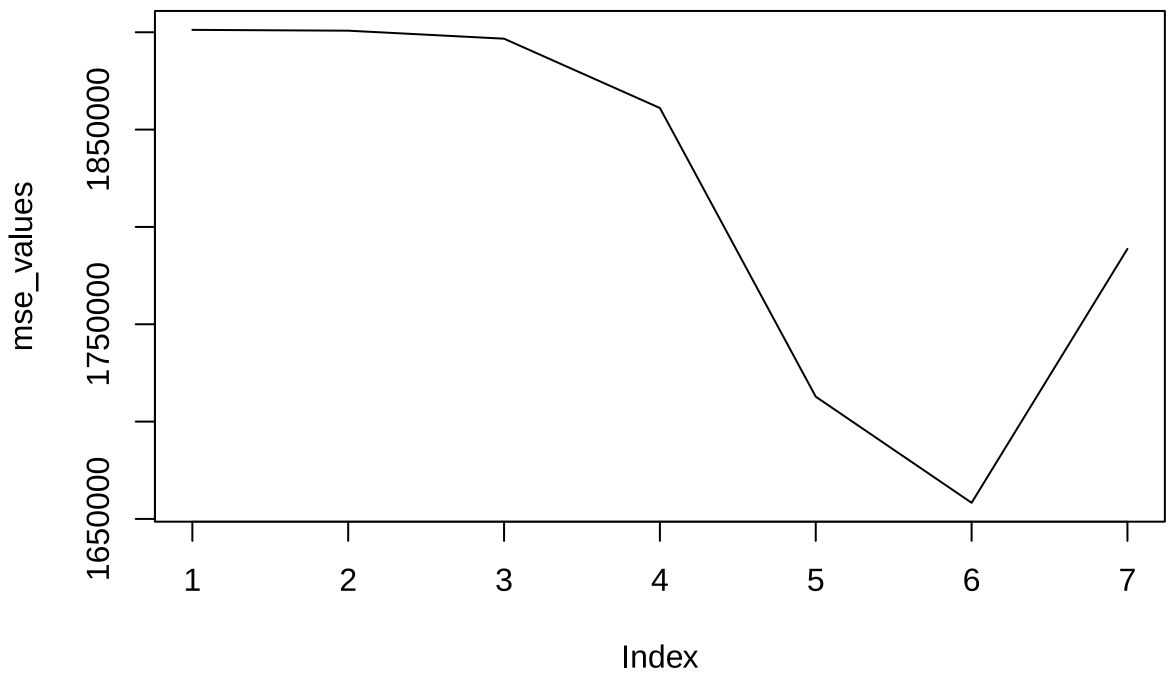
OLS is scale equivariant, so there is a direct correlation between a X and the beta coeff. If you where to multiply X_i with a constant c, the beta coeff will be directly reduced by 1/c.

However then we are working with ridge and lasso regression the model is sensitive to size of the beta coeff. This is caused by the shrinkage penalty since it does not have the same interaction with the beta coeff and X as a OLS. There is also a possibility that the beta coeff to a given predictor in lasso and rigde regression might even be affected by other predictors scaling. Therefore one need to scale and center the predictors to get a correct penalty. The X demean is a way to center the variable, however not as good as a fully scaled X.

```r
la_index <- c(0.1, 1, 10, 100, 1000, 10000, 100000)

mse_values <- sapply(la_index, function(lambda){
  loo(la = lambda, X, y)
})

min_mse <- which.min(mse_values)
la_index[min_mse]
```

```
## [1] 10000
```

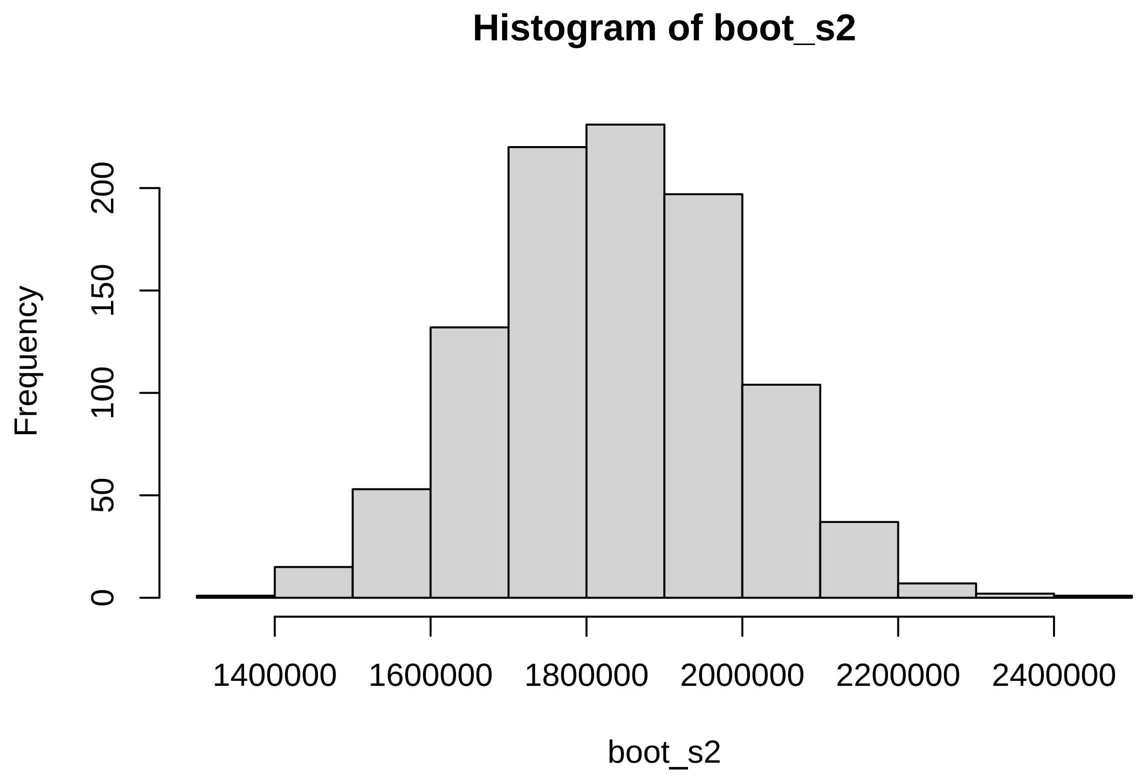
```r
plot(mse_values, type = "l")
```



The optimal “la” seems to be somewhere around 10 000.

d)

```r
set.seed(123)

boot <- function(x, n_rep) {
  s2_values <- numeric(n_rep)

  for (i in 1:n_rep) {
    n <- length(x)

    values <- sample(x, n, replace = TRUE)
    m <- mean(values)

    s2_values[i] <- (1/(n - 1))* sum((values - m)^2)
  }

  return(s2_values)
}


boot_s2 <- boot(y, 1000)

hist(boot_s2)
```



```
low_boot_s2 <- quantile(boot_s2, 0.025)
high_boot_s2 <- quantile(boot_s2, 0.975)


cbind(low_boot_s2, high_boot_s2)
```

```
##      low_boot_s2 high_boot_s2
## 2.5%     1516579      2137712
```

```
sd(y)^2
```

```
## [1] 1846096
```

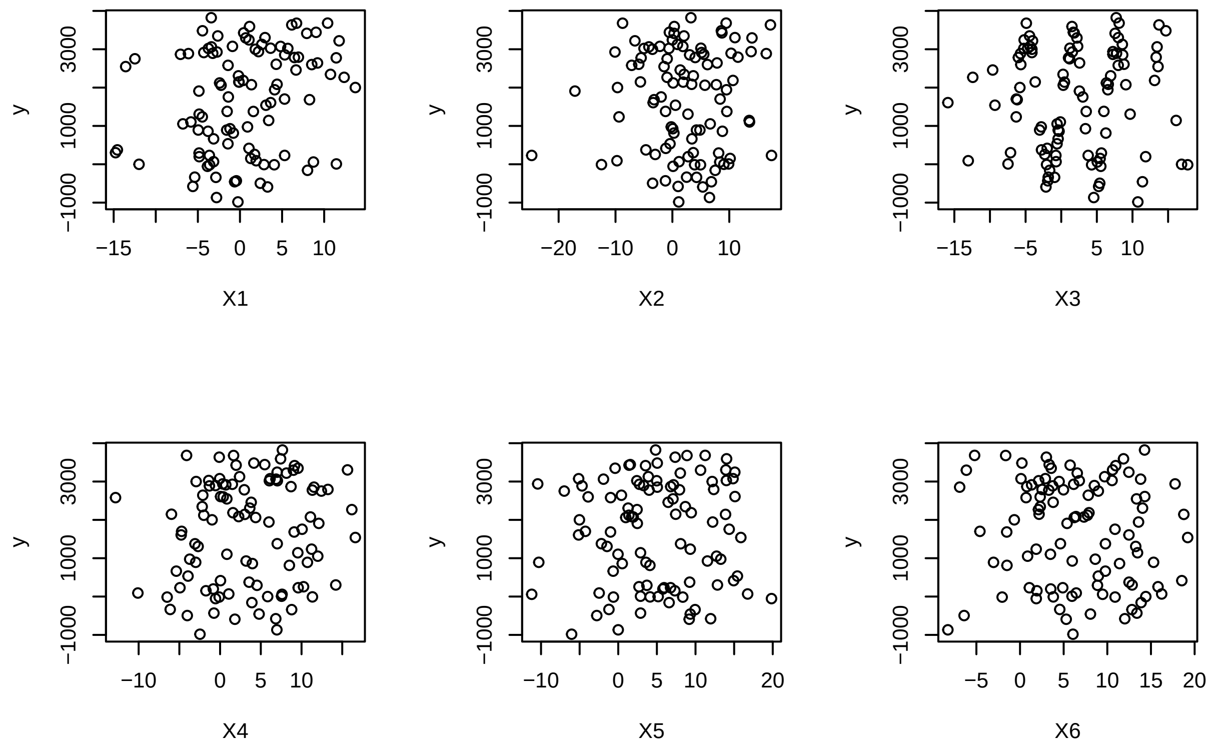
The bootstrap method give a 95% confidence interval of the variance of y to be (1516579, 2137712). This seems to be correct taking into account that the Var(y) is 1846096, right between the lower and higher end of the confidence interval.

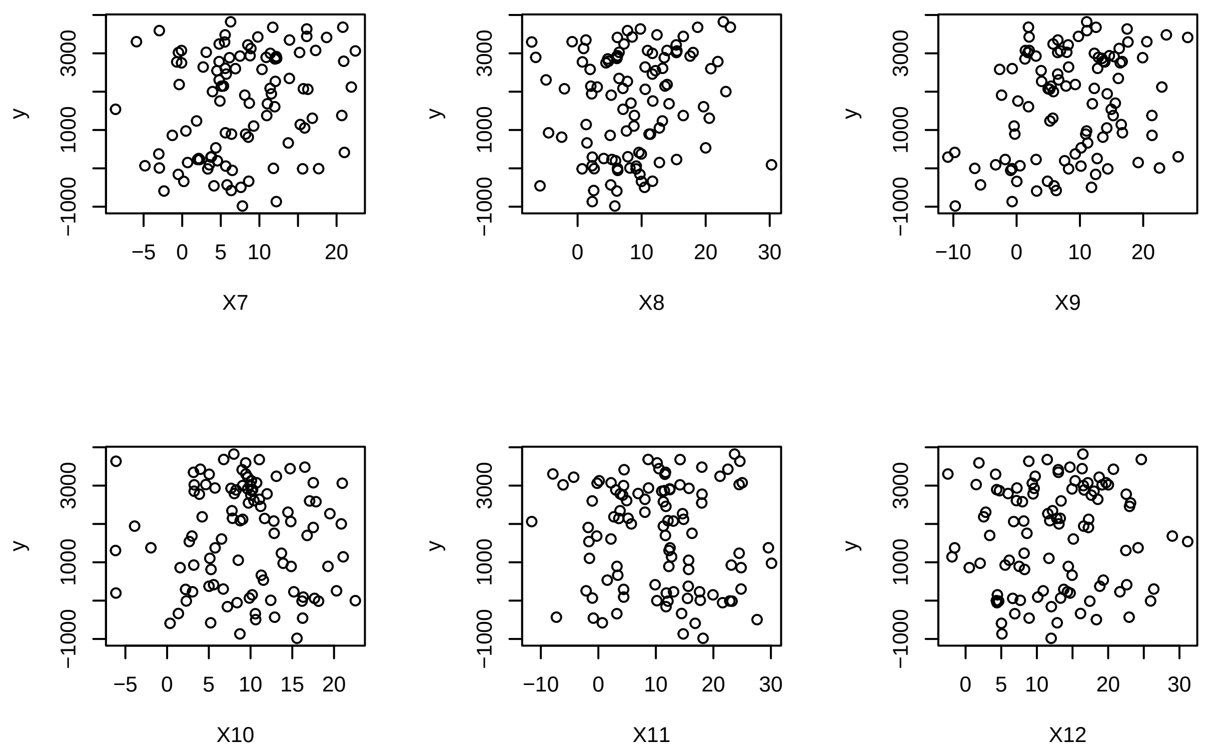
e)

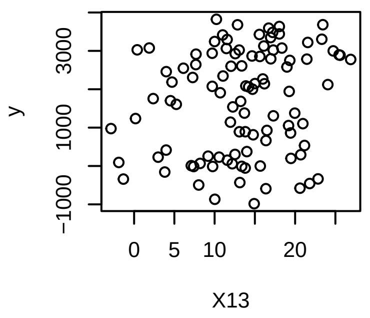
Since the task asks for MSE training results, I will work with the whole dataset. There is no need to have out of sample data.

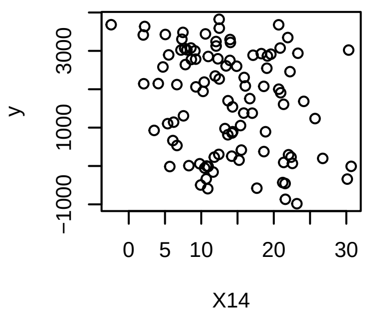
First we are looking for variables that do not have a linear relationships to y. We do that by plotting the different predictors against y.

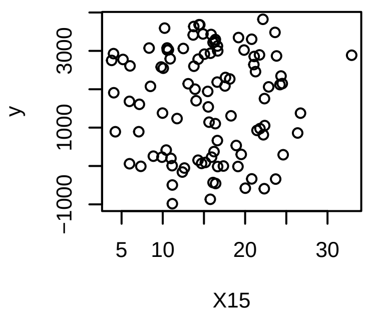
```
par(mfrow = c(2,3))

plot(y ~ ., data = Xy)
```

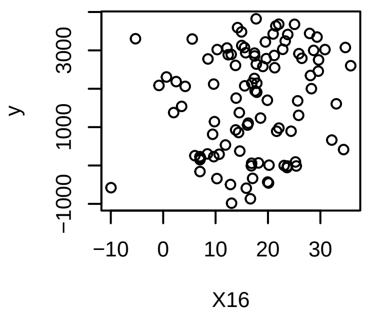


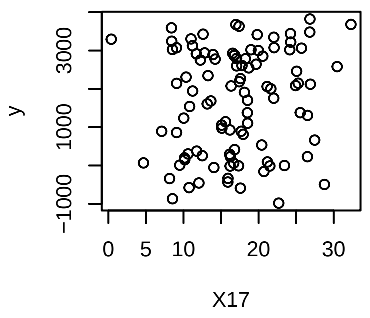

y
−1000
1000
3000
X7
−5
0
5
10
20
X8
0
10
20
30
X9
−10
0
10
20
X10
−5
0
5
10
15
20
X11
−10
0
10
20
30
X12
0
5
10
20
30

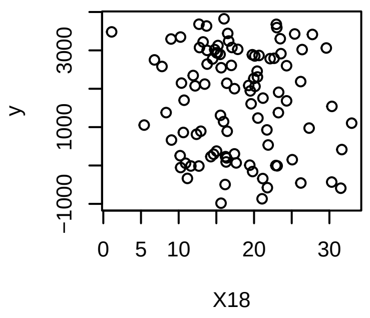

y
−1000
1000
3000
0
5
10
20
X13

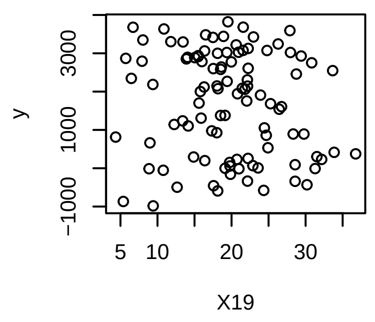

y
−1000
1000
3000
0
5
10
20
30
X14

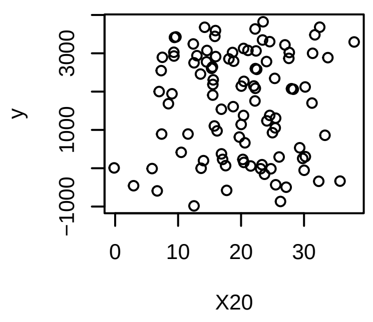

y
−1000
1000
3000
5
10
20
30
X15


y
−1000
1000
3000
−10
0
10
20
30
X16


y
−1000
1000
3000
0
5
10
20
30
X17

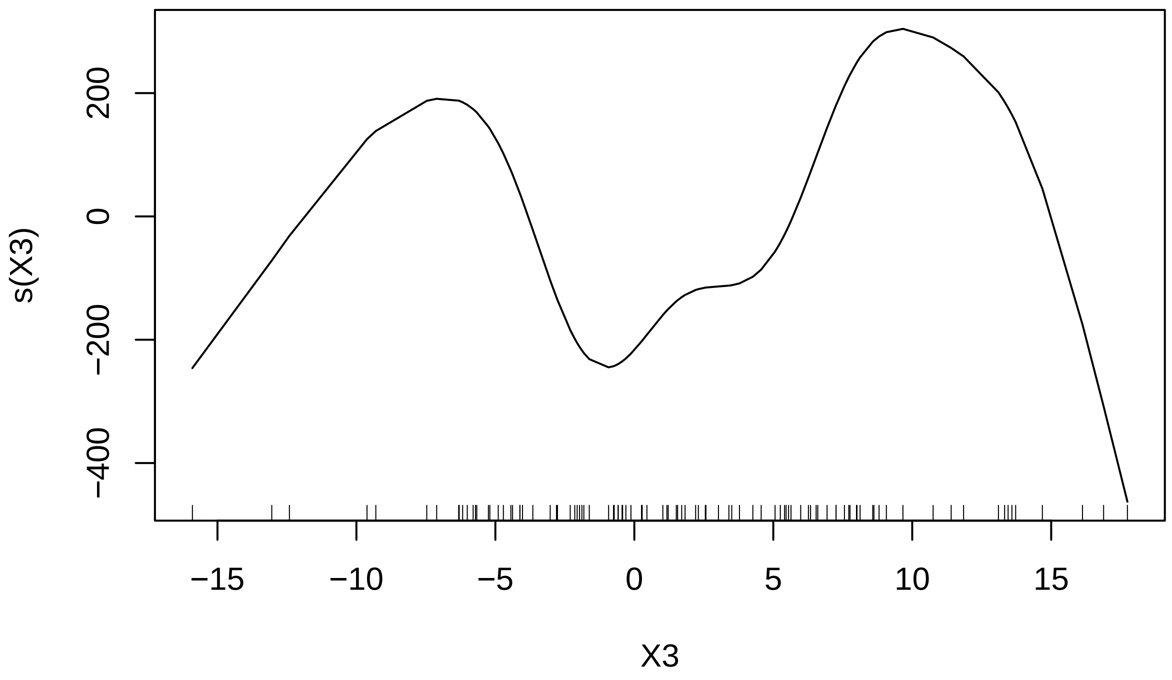

y
−1000
1000
3000
0
5
10
20
30
X18





There do not seem to any clear links between X and y. X3 seems somewhat promising since it seems to be a sinus function. We will try that.

```r
library(gam)

g3 <- gam(y ~ s(X3) , data = Xy)

plot(g3)
```



```r
summary(g3)
```

```
##
## Call: gam(formula = y ~ s(X3), data = Xy)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -2954.9 -1214.9   179.2  1169.2  2048.4
##
## (Dispersion Parameter for gaussian family taken to be 1747288)
##
##     Null Deviance: 182763489 on 99 degrees of freedom
## Residual Deviance: 165992483 on 95 degrees of freedom
## AIC: 1728.016
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##           Df    Sum Sq Mean Sq F value Pr(>F)
## s(X3)      1    119368  119368  0.0683 0.7944
## Residuals 95 165992483 1747288
##
## Anova for Nonparametric Effects
##             Npar Df Npar F   Pr(F)
## (Intercept)
## s(X3)             3 3.1767 0.02762 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
pred_g3 <- predict(g3)

mse_g3 <- mean((y - pred_g3)^2)

pred_lm <- predict(model_lm)

mse_lm <-  mean((y - pred_lm)^2)

cbind(mse_lm, mse_g3)
```

```
##       mse_lm  mse_g3
## [1,] 1207608 1659925
```

The g3 model do preform bad compared to the lm. Lets try it with all the other predictors, when they are linear.

```
gl3 <- gam(y ~ . -X3 + s(X3), data = Xy)

pred_gl3 <- predict(gl3)

mse_gl3 <- mean((y - pred_gl3)^2)

cbind(mse_lm, mse_g3, mse_gl3)
```

```
##       mse_lm  mse_g3 mse_gl3
## [1,] 1207608 1659925 1101079
```

Now “gl3” preforms better than the linear model.

Lets look at the summary of “model_lm” to see if some of the predictors have high p-values. Maybe a model are better if we make them a include them as s() in gam.

```
summary(model_lm)
```

```
##
## Call:
## lm(formula = y ~ X)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2532.62  -872.31    48.46   829.86  2168.58
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2251.656   1636.230   1.376   0.1727
## XX1           26.198     26.527   0.988   0.3264
## XX2          -17.161     26.477  -0.648   0.5188
## XX3           21.793     29.151   0.748   0.4569
## XX4           61.559     25.273   2.436   0.0171 *
## XX5            8.023     24.744   0.324   0.7466
## XX6          -18.217     26.218  -0.695   0.4892
## XX7           24.787     25.743   0.963   0.3385
## XX8           51.744     24.789   2.087   0.0401 *
## XX9           35.770     19.672   1.818   0.0728 .
## XX10         -26.240     26.988  -0.972   0.3339
## XX11         -32.985     18.154  -1.817   0.0730 .
## XX12         -12.264     25.416  -0.483   0.6308
## XX13           3.535     25.355   0.139   0.8895
## XX14         -43.239     27.344  -1.581   0.1178
## XX15          26.127     31.098   0.840   0.4034
## XX16          32.451     20.651   1.571   0.1201
## XX17         -10.731     37.787  -0.284   0.7772
## XX18         -41.534     26.984  -1.539   0.1278
## XX19           3.477     29.366   0.118   0.9061
## XX20         -24.029     22.884  -1.050   0.2969
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1236 on 79 degrees of freedom
## Multiple R-squared:  0.3393, Adjusted R-squared:  0.172
## F-statistic: 2.028 on 20 and 79 DF,  p-value: 0.01434
```

I choose all variables with higher than 0.5 p-value.

```
gsome <- gam(y ~ . + s(X2) + s(X6) + s(X12) + s(X13) + s(X17) + s(X19)
           -X2 -X6 -X12 -X13 -X17 -X19, data = Xy)

pred_gsome <- predict(gsome)

mse_gsome <- mean((y - pred_gsome)^2)

cbind(mse_lm, mse_g3, mse_gl3, mse_gsome)
```

```
##       mse_lm  mse_g3 mse_gl3 mse_gsome
## [1,] 1207608 1659925 1101079  909596.5
```

Lets try every predictor since, no predictor do have a true linear relationships with y.

```
library(gam)

gall <- gam(y ~
               s(X1) + s(X2) + s(X3) + s(X4) + s(X5) + s(X6) + s(X7) +
               s(X8) + s(X9) + s(X10) + s(X11) + s(X12) + s(X13) +
               s(X14) + s(X15) + s(X16) + s(X17) + s(X18) + s(X19) +
               s(X20), data = Xy)


pred_gall <- predict(gall)

mse_gall <- mean((y - pred_gall)^2)

cbind(mse_lm, mse_g3,  mse_gl3, mse_gsome, mse_gall)
```

```
##       mse_lm  mse_g3 mse_gl3 mse_gsome mse_gall
## [1,] 1207608 1659925 1101079  909596.5 370283.2
```

"gall" preforms much better than the other models.

To calculate the Rˆ2 we need some basic knowledge.

Number 1: MSE = RSS/n

Number 2: Rˆ2 = 1 - (RSS/TSS)

Number 3: TSS = the sum of (y_i - mean of y)ˆ2

```
mean_y <- mean(y)
tss <- sum((y - mean_y)^2)
n <- length(y)

r2_lm <- 1 - (mse_lm * n/tss)
model_lm_s <- summary(model_lm)
cbind(r2_lm, model_lm_s$r.squared)
```

```
##          r2_lm
## [1,] 0.3392508 0.3392508
```

```
r2_g3 <- 1 - (mse_g3 * n/tss)

r2_gl3 <- 1 - (mse_gl3 * n/tss)

r2_gsome <- 1 - (mse_gsome * n/tss)

r2_gall <- 1 - (mse_gall * n/tss)

cbind(r2_lm, r2_g3, r2_gl3, r2_gsome, r2_gall)
```

```
##          r2_lm      r2_g3    r2_gl3  r2_gsome   r2_gall
## [1,] 0.3392508 0.09176344 0.3975391 0.5023095 0.7973976
```

The gam model with all predictors as a gam is the best preforming. With a Rˆ2 equal to ca 80%. However this is not a adj Rˆ2, so we lack the penalty for many predictors, so the adj Rˆ2 might be much lower.

There is a strong possibility that "gall" is very overfit with low bias and high variance. We have done no work to validate that it is a good prediction model.

Task 2

a)

```
library(insuranceData)

data("dataOhlsson")

ydata <- dataOhlsson
```

```r
ydata$claim <- ydata$antskad >= 1

str(ydata)
```

```
## 'data.frame':    64548 obs. of  10 variables:
##  $ agarald : int  0 4 5 5 6 9 9 9 10 10 ...
##  $ kon     : Factor w/ 2 levels "K","M": 2 2 1 1 1 1 1 2 2 2 ...
##  $ zon     : int  1 3 3 4 2 3 4 4 2 4 ...
##  $ mcklass : int  4 6 3 1 1 3 3 4 3 2 ...
##  $ fordald : int  12 9 18 25 26 8 6 20 16 17 ...
##  $ bonuskl : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ duration: num  0.175 0 0.455 0.173 0.181 ...
##  $ antskad : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ skadkost: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ claim   : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
```

```r
factor_columns <- c("kon", "zon", "mcklass", "bonuskl")
ydata[factor_columns] <- lapply(ydata[factor_columns], as.factor)
```

"zon" is defined as a factor since it is a class on Swedish regions.

"mcklass" is the classification of different "EV ratios" that takes into account the ration between the power of MC and the weight of the MC.

"bonuskl" is defines as a factor since it represent what class of bouns insurance scheme they are a part of. The longer without accidents, the higher value.

```r
ydata$antskad <- NULL
ydata$skadkost <- NULL
```

I have to remove "antskad" and "skadkost" from the data since they are directly linked to the "claim" variable.

I argue for keeping the "bonuskl" predictor since it is more a measure of how long one have been without a claim rather than a metric on if a claim have happend.

```r
table(ydata$claim)
```

```
##
## FALSE  TRUE
## 63878   670
```

```r
sum(ydata$claim / nrow(ydata))
```

```
## [1] 0.01037987
```

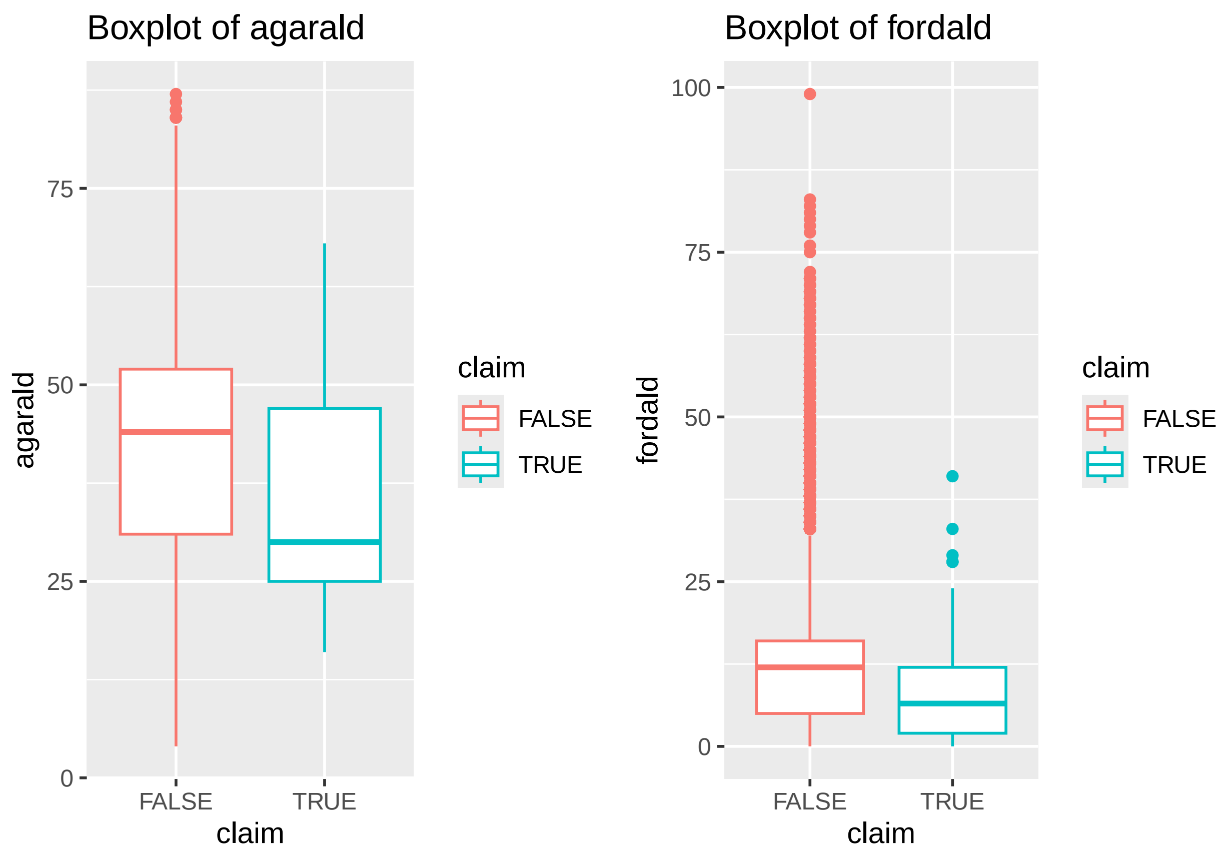
It is interesting that there is such a big skew to False. Around 1% have had a claim.

I split the data since the task clearly states that you should do that.

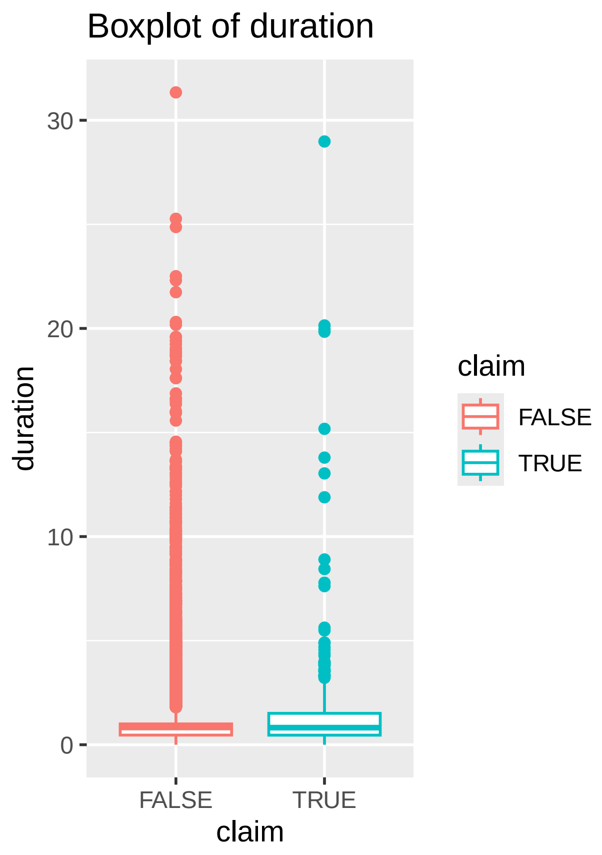
```
set.seed(123)

ind <- sample(1:nrow(ydata), size = floor(nrow(ydata) / 2))

train <- ydata[ind, ]
test <- ydata[-ind, ]
```

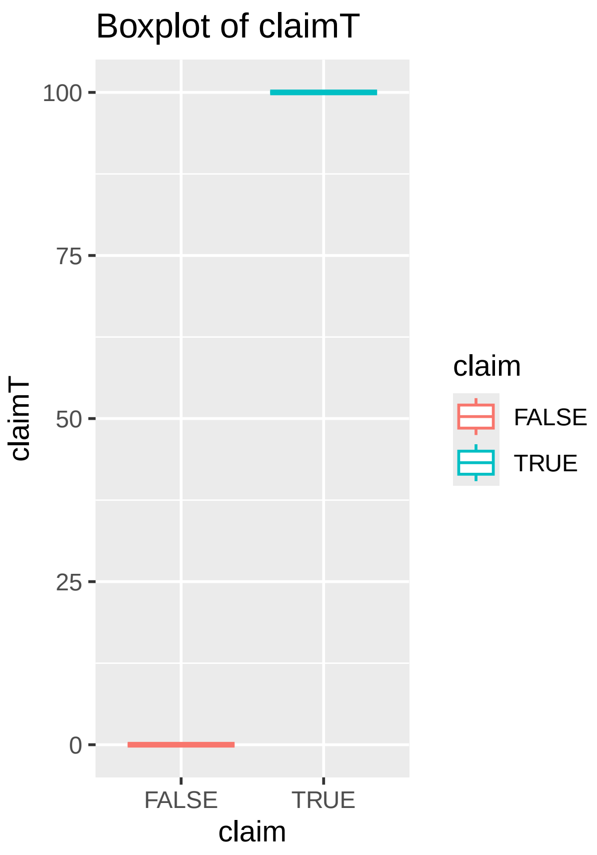
```
# This is a fast aggregation of plots and summaries of
# the predictors relationships to "claim"

train <- train %>%
  mutate(claimT = ifelse(claim == T, 100, 0),
         # Using 100 here such that I get numbers in "%" in s_list.
         claim = as.factor(claim))

numeric_cols <- sapply(train, is.numeric)
numeric_cols["calimT"] <- F

factor_cols <- sapply(train, function(col) is.factor(col) || is.character(col))
factor_cols["claim"] <- F


plot_list <- list()

# Boxplots for numeric columns
for (colname in names(train)[numeric_cols]) {
  p <- ggplot(train, aes(x = claim, y = .data[[colname]], color = claim)) +
  geom_boxplot() +
  labs(title = paste("Boxplot of", colname), x = "claim", y = colname)

  plot_list[[length(plot_list) + 1]] <- p
}


library(gridExtra)
library(grid)

grid_list <- list()

for (i in seq(1, length(plot_list), by = 2)) {
  grid_obj <- arrangeGrob(
  grobs = plot_list[i:min(i + 1, length(plot_list))],
  ncol = 2
)
  grid_list[[length(grid_list) + 1]] <- grid_obj
}

s_list <- list()

for (colname in names(train)[factor_cols]) {
  s_list[[colname]] <- tapply(train$claimT, train[[colname]], mean, na.rm = T)
}
```

```r
grid.newpage()
grid.draw(grid_list[[1]])
```



```r
grid.newpage()
grid.draw(grid_list[[2]])
```





```
s_list
```

```
## $kon
##        K        M
## 0.573418 1.117155
##
## $zon
##         1         2         3         4         5         6         7
## 2.0469877 1.4666217 0.9601763 0.6835545 0.3439381 0.4173187 0.5714286
##
## $mcklass
##         1         2         3         4         5         6         7
## 0.5772006 1.1265490 0.8728573 0.7736944 1.1527873 1.9426676 1.0000000
##
## $bonuskl
##         1         2         3         4         5         6         7
## 0.9511993 0.6451613 0.9321293 1.0024196 0.7773028 0.7108118 1.5190439
```

Based on the plots and the average amount of claims in each category in the factor columns where the data is presented in %, there is no clear predictor that should not be included. All predictor seems useful, for example the sex, there the male is ca 100% more likely to have a claim than a female.

b)

```r
train$claimT <- NULL

model_log_all <- glm(claim ~ ., data = train, family = binomial())

summary(model_log_all)
```

```
##
## Call:
## glm(formula = claim ~ ., family = binomial(), data = train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.305767   0.362613  -6.359 2.03e-10 ***
## agarald     -0.047866   0.005008  -9.558  < 2e-16 ***
## konM         0.676923   0.200700   3.373 0.000744 ***
## zon2        -0.281711   0.154657  -1.822 0.068528 .
## zon3        -0.786883   0.170424  -4.617 3.89e-06 ***
## zon4        -1.150491   0.164125  -7.010 2.39e-12 ***
## zon5        -1.877668   0.514098  -3.652 0.000260 ***
## zon6        -1.464143   0.372702  -3.928 8.55e-05 ***
## zon7        -1.283589   1.013377  -1.267 0.205282
## mcklass2     0.398606   0.292508   1.363 0.172971
## mcklass3    -0.046144   0.255342  -0.181 0.856592
## mcklass4    -0.095665   0.269807  -0.355 0.722914
## mcklass5     0.249489   0.259756   0.960 0.336816
## mcklass6     0.704384   0.254898   2.763 0.005720 **
## mcklass7    -0.306640   0.556089  -0.551 0.581344
## fordald     -0.079741   0.009378  -8.503  < 2e-16 ***
## bonuskl2    -0.350021   0.223444  -1.566 0.117236
## bonuskl3     0.010843   0.216963   0.050 0.960141
## bonuskl4     0.005628   0.225679   0.025 0.980105
## bonuskl5    -0.254256   0.258108  -0.985 0.324586
## bonuskl6    -0.316552   0.263901  -1.200 0.230329
## bonuskl7     0.413306   0.161584   2.558 0.010533 *
## duration     0.181016   0.021950   8.247  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3717.9  on 32273  degrees of freedom
## Residual deviance: 3313.7  on 32251  degrees of freedom
## AIC: 3359.7
##
## Number of Fisher Scoring iterations: 8
```

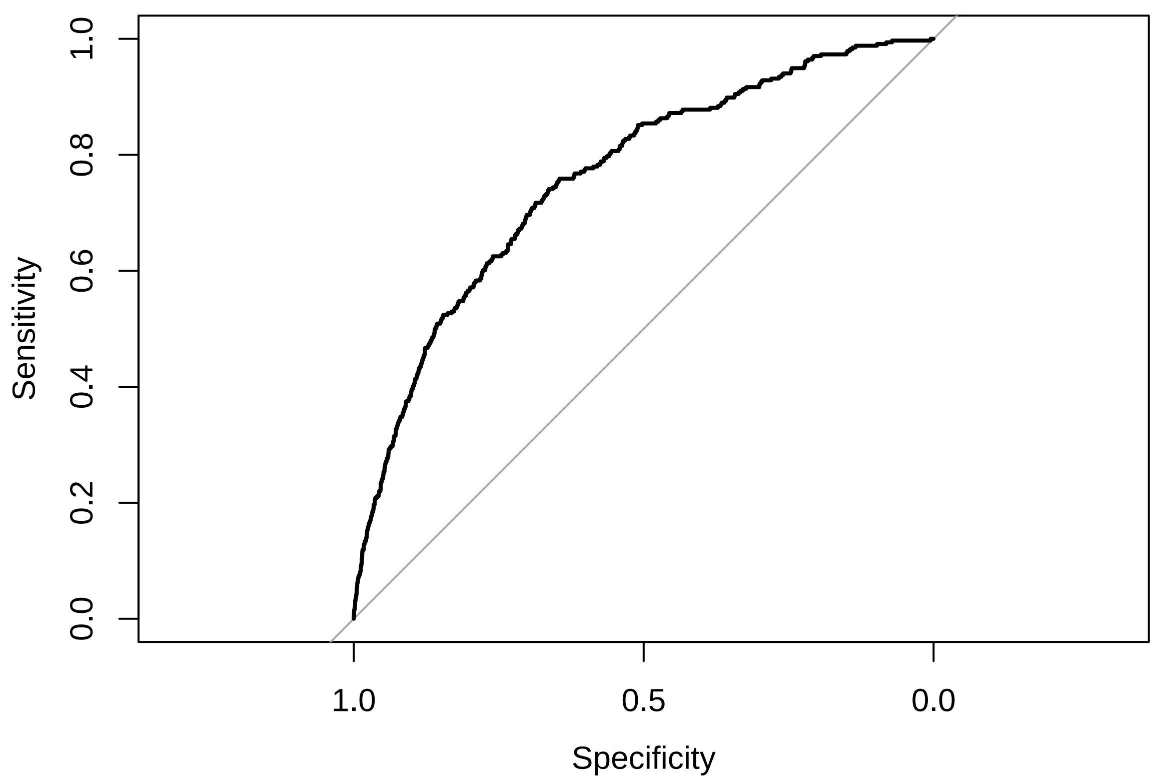
Based on the knowledge of the logistic regression with all predictors, I now remove "bonuskl" and "mcklass". Those where the least helpful in the last model.

```r
model_log_some <- glm(claim ~ . -bonuskl -mcklass,
                      data = train,
                      family = binomial())

summary(model_log_some)
```

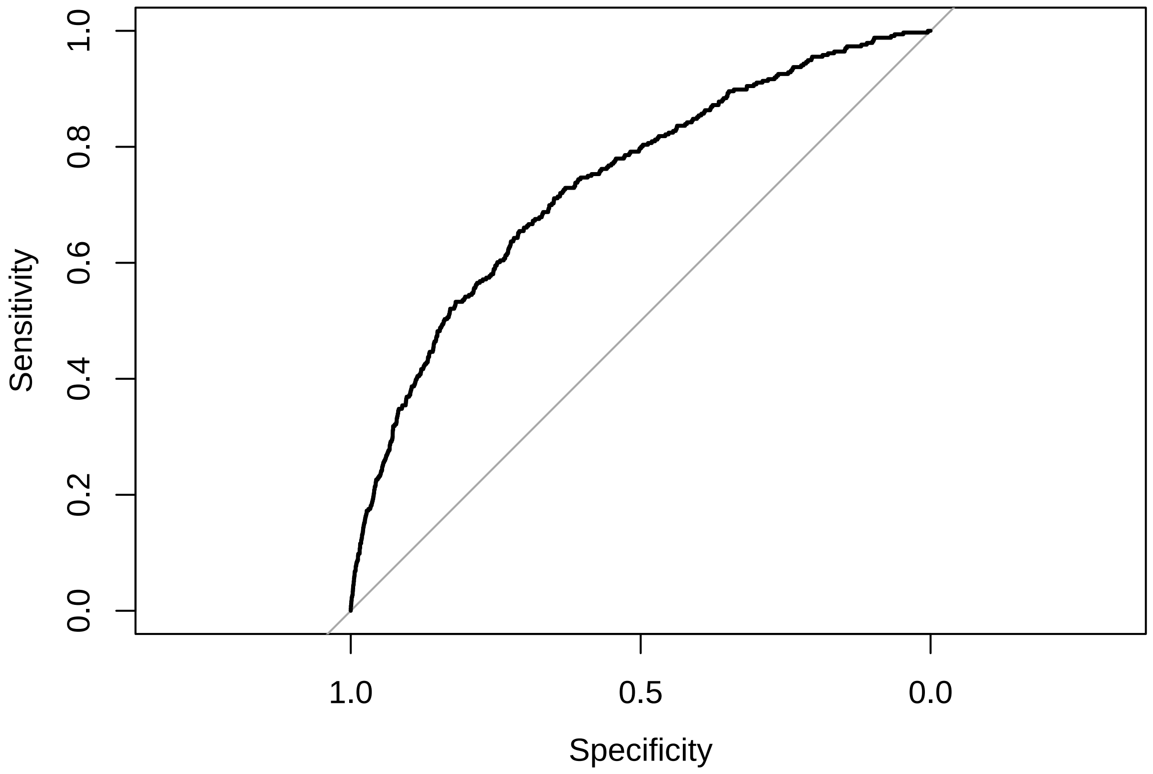
```
##
## Call:
## glm(formula = claim ~ . - bonuskl - mcklass, family = binomial(),
##     data = train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.229932   0.271093  -8.226  < 2e-16 ***
## agarald     -0.045544   0.004729  -9.631  < 2e-16 ***
## konM         0.692202   0.199399   3.471 0.000518 ***
## zon2        -0.250733   0.153682  -1.631 0.102785
## zon3        -0.732927   0.168951  -4.338 1.44e-05 ***
## zon4        -1.131166   0.161665  -6.997 2.62e-12 ***
## zon5        -1.754933   0.513016  -3.421 0.000624 ***
## zon6        -1.366306   0.371346  -3.679 0.000234 ***
## zon7        -1.104859   1.011557  -1.092 0.274730
## fordald     -0.078598   0.009020  -8.714  < 2e-16 ***
## duration     0.206305   0.020626  10.002  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3717.9  on 32273  degrees of freedom
## Residual deviance: 3371.5  on 32263  degrees of freedom
## AIC: 3393.5
##
## Number of Fisher Scoring iterations: 8
```

c)

```r
library(pROC)

prob_all <- predict(model_log_all, newdata = test, type = "r")

roc_all <- roc(test$claim, prob_all)

plot(roc_all)
```



```r
auc_all <- auc(roc_all)


prob_some <- predict(model_log_some, newdata = test, type = "r")

roc_some <- roc(test$claim, prob_some)

plot(roc_some)
```



```r
auc_some <- auc(roc_some)

cbind(auc_all, auc_some)
```

```
##        auc_all  auc_some
## [1,] 0.7600109 0.7364729
```

"model_log_all" preforms better than "model_log_some" with the metric area under curve(AUC).

Based on the ROC for both "all" and "some" logistic regression there is no clear threshold to choose. One could try to optimize based on accuracy, however that will skew very hard against "False" since 99% of all obs are that, so that will cause the model to have a threshold close to 1.

The question more about what do we want to predict. Do we want to be cautions, such that our true positive is high together with high false positive, or do we want high true negative with all actual true positive classified as negative?

I think we rather want a high true positive, than a low one, because that is where the risk is for these insurance companies. They already know that 99% of the customers do not claim the insurance.

```r
pred_all <- prob_all > 0.5
prop.table(table(test$claim, pred_all), margin = 1)
```

```
##        pred_all
##                FALSE         TRUE
##   FALSE 9.999374e-01 6.262133e-05
##   TRUE  9.970238e-01 2.976190e-03
```

50% is a terrible threshold at true positive.

```r
pred_all <- prob_all > 0.1
prop.table(table(test$claim, pred_all), margin = 1)
```

```
##        pred_all
##               FALSE        TRUE
##   FALSE 0.997964807 0.002035193
##   TRUE  0.979166667 0.020833333
```

```r
pred_all <- prob_all > 0.01
table(test$claim, pred_all)
```

```
##        pred_all
##         FALSE  TRUE
##   FALSE 21915 10023
##   TRUE     95   241
```

```r
prop.table(table(test$claim, pred_all), margin = 1)
```

```
##        pred_all
##             FALSE      TRUE
##   FALSE 0.6861732 0.3138268
##   TRUE  0.2827381 0.7172619
```

Threshold 0.01 seems to be quite good, however very many is placed in the false positive bucket. Let's try some numbers between 0.01 and 0.1

```r
pred_all <- prob_all > 0.05
table(test$claim, pred_all)
```

```
##        pred_all
##         FALSE  TRUE
##   FALSE 31222   716
##   TRUE    288    48
```

```r
pred_all <- prob_all > 0.02
table(test$claim, pred_all)
```

```
##        pred_all
##         FALSE  TRUE
##   FALSE 27341  4597
##   TRUE    165   171
```

```r
prop.table(table(test$claim, pred_all), margin = 1)
```

```
##        pred_all
##             FALSE      TRUE
##   FALSE 0.8560649 0.1439351
##   TRUE  0.4910714 0.5089286
```

0.02 seems to be a good threshold, this at least have identified 50% of the actual positive cases.

```
pred_some <- prob_some > 0.02

table(test$claim, pred_all)
```

```
##        pred_all
##         FALSE  TRUE
##   FALSE 27341  4597
##   TRUE    165   171
```

```
prop.table(table(test$claim, pred_all), margin = 1)
```

```
##        pred_all
##             FALSE      TRUE
##   FALSE 0.8560649 0.1439351
##   TRUE  0.4910714 0.5089286
```

As a metric for comparing the different models I will use the AUC value gotten from the ROC. The reason for this is that the AUC takes into account the different threshold levels and we have a trade off between true positive rate and true negative rate. Since both are "Rates" the absolute number of False is not a problem.

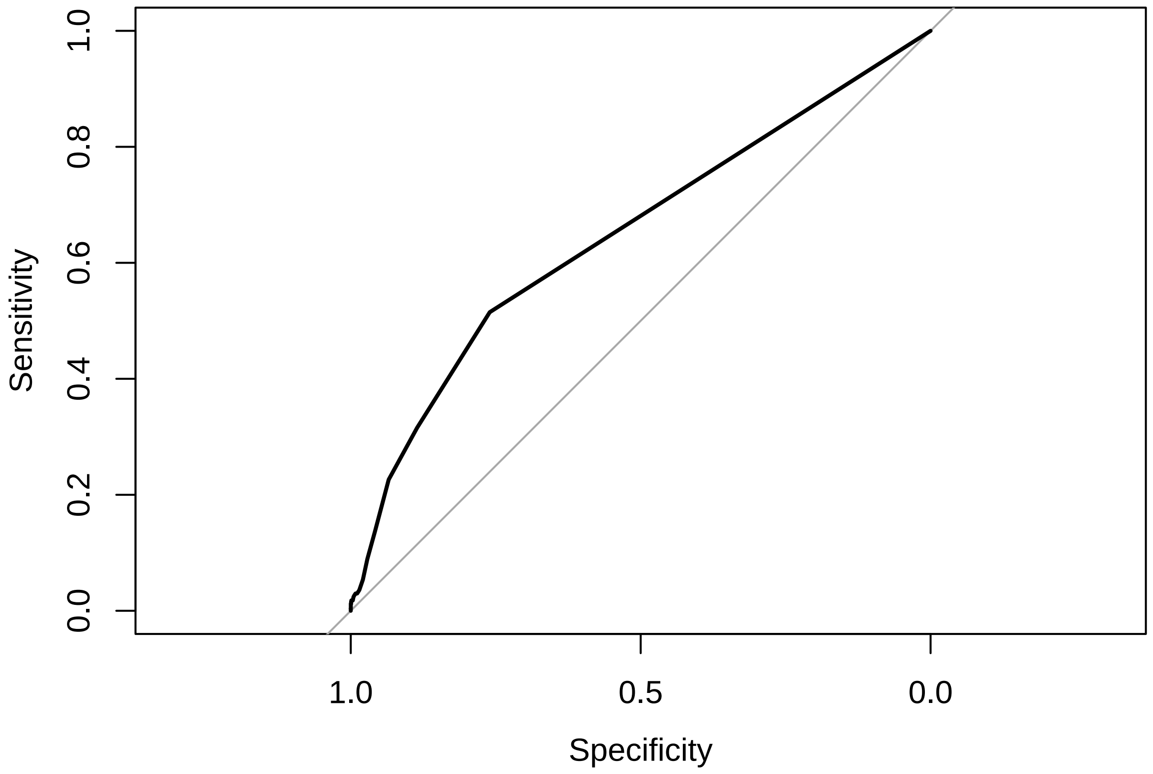
```
cbind(auc_all, auc_some)
```

```
##        auc_all  auc_some
## [1,] 0.7600109 0.7364729
```

The logistic regression with all predictors seems the best. However they where equally good at the confusion matrix with the threshold 0.02

d)

```
library(randomForest)

model_rf <- randomForest(claim ~ ., data = train, ntree = 50, mtry = 3)

prob_rf <- predict(model_rf, newdata = test, type = "prob")

prob_rf <- prob_rf[,2]

rf <- roc(test$claim, prob_rf)

plot(rf)
```



```r
auc_rf <- auc(rf)

cbind(auc_all, auc_some, auc_rf)
```

```
##        auc_all  auc_some    auc_rf
## [1,] 0.7600109 0.7364729 0.6481608
```

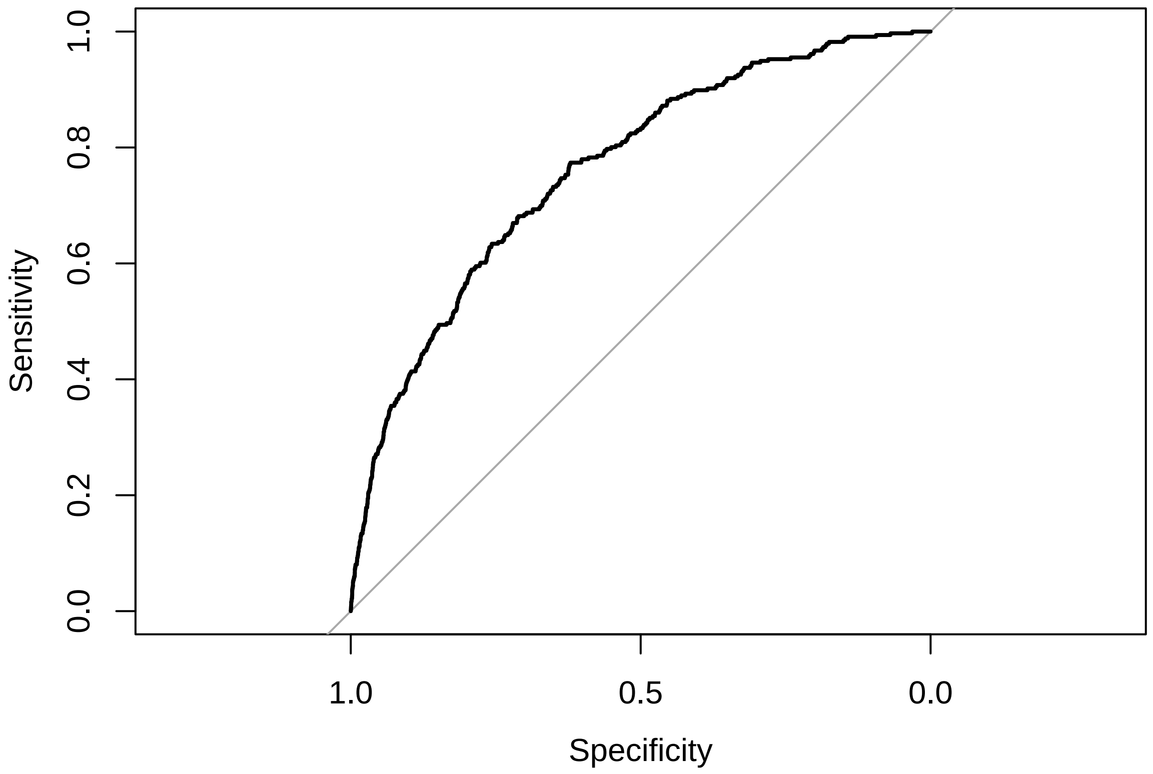
Random forest did worse than the logistic regression.

Ntree is 50 to keep the code faster. One could use cross validation to find an optimal mtry, mtry = 3 might not be optimal.

A note, the ROC plot seems very strange, it is a straight line from the middle and out.

e)

```r
library(gbm)
train$claim1 <- ifelse(train$claim == T, 1, 0)

model_boost <- gbm(claim1 ~ . - claim,
                   data = train,
                   distribution = "bernoulli",
                   n.trees = 500,
                   interaction.depth = 4,
                    shrinkage = 0.01
                     )

prob_boost <- predict(model_boost, newdata = test, n.trees = 500,
                      type = "response")

boost <- roc(test$claim, prob_boost)

plot(boost)
```



```r
auc_boost <- auc(boost)

cbind(auc_all, auc_some, auc_rf, auc_boost)
```

```
##        auc_all  auc_some    auc_rf auc_boost
## [1,] 0.7600109 0.7364729 0.6481608 0.7625533
```

With the AUC value as the metric the boosting model did the best, just beating the logistic regression with all predictors.

You could use cross validation with different n.trees, interaction.depth, shrinkage to optimize the boosting further, that might increase the AUC value.

NB: We cannot conclude that boosting is better logistic regression since they are very close and we have some randomness included, for example through set.seed in the split. A another split might cause another model to be better, and I would expect the models to quite sensitive to the distribution of Claim == TRUE in the training and test data.