Final Exam

Sandra Villamar

2023-03-16

Load Data and Libraries

```
dat = read.csv("stackloss.csv", sep=" ")

library(car)
```

```
## Loading required package: carData
```

```
library(leaps)
library(lars)
```

```
## Loaded lars 1.3
```

Problem 1A

```
m = lm(Stack.Loss ~ Acid.Conc, data = dat)
summary(m)
```

```
##
## Call:
## lm(formula = Stack.Loss ~ Acid.Conc, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1584 -0.5584 -0.3066  0.1247  2.2416
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -42.7441    23.4027  -1.826   0.0835 .
## Acid.Conc     0.7590     0.3992   1.901   0.0725 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9565 on 19 degrees of freedom
## Multiple R-squared:  0.1599, Adjusted R-squared:  0.1156
## F-statistic: 3.615 on 1 and 19 DF,  p-value: 0.07252
```

The estimated value of the intercept is -42.7441. This means that when the percentage concentration of circulating acid is 0%, the model predicts that the percentage of the ingoing ammonia lost in the process of the reaction from ammonia to nitric acid is about -42.7%. Hence at zero acid concentration, the reaction from ammonia to nitric acid actually produces 42.7% more ammonia instead of losing it. This obviously does not make sense. The reason why we get this value is because when building the model, we only use acid concentrations between 57.2% and 59.3%. Therefore looking at the model's behavior at 0% is extrapolating from our given information and leads to false conclusions. We can fix this by standardizing our data.

Furthermore, we have the estimated value of the slope at 0.7590. This means that with a 1% increase in the concentration of circulating acid, the model predicts a 0.7590% increase in the percentage of lost ammonia.

Problem 1B

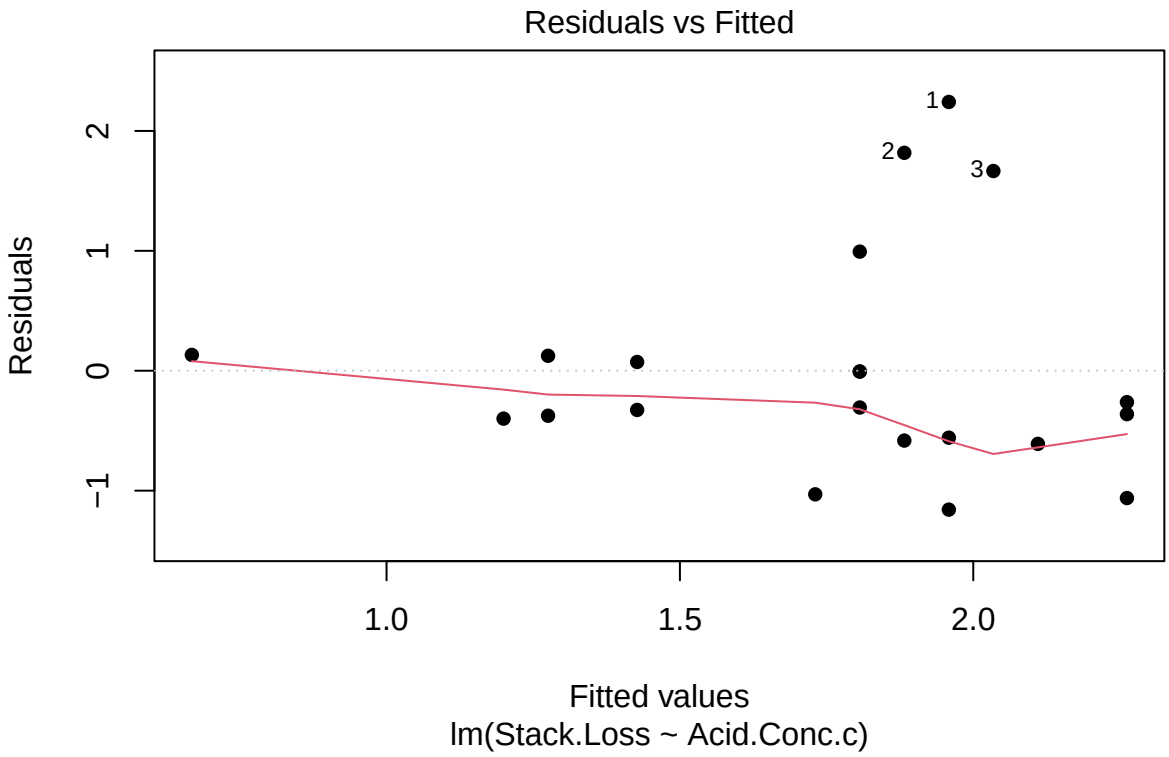
```
Acid.Conc.c = dat$Acid.Conc - mean(dat$Acid.Conc)
OLS1 = lm(Stack.Loss ~ Acid.Conc.c, data = dat)
summary(OLS1)
```

```
##
## Call:
## lm(formula = Stack.Loss ~ Acid.Conc.c, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1584 -0.5584 -0.3066  0.1247  2.2416
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.7524     0.2087   8.395 8.13e-08 ***
## Acid.Conc.c   0.7590     0.3992   1.901   0.0725 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9565 on 19 degrees of freedom
## Multiple R-squared:  0.1599, Adjusted R-squared:  0.1156
## F-statistic: 3.615 on 1 and 19 DF,  p-value: 0.07252
```

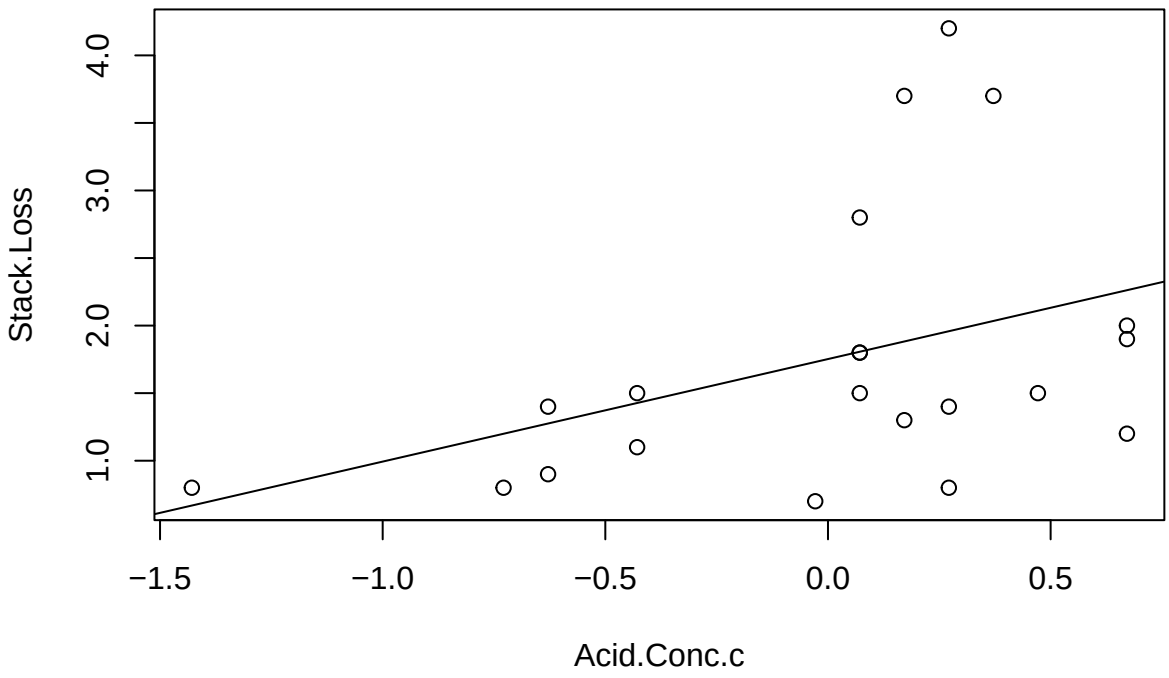
With the centered acid concentration, we get the same slope value as the un-centered (which makes sense as we simply shifted the predictor variable). Now though, we get an intercept value that makes more sense because the predictor value of zero is within the range of the model input. The intercept interpretation is that at the mean acid concentration (~58.63%), the model predicts an ammonia loss of 1.75%.

Problem 1C

```
# 1. errors have zero mean
plot(OLS1, which=1, pch=16)  # residuals vs. fitted values
```



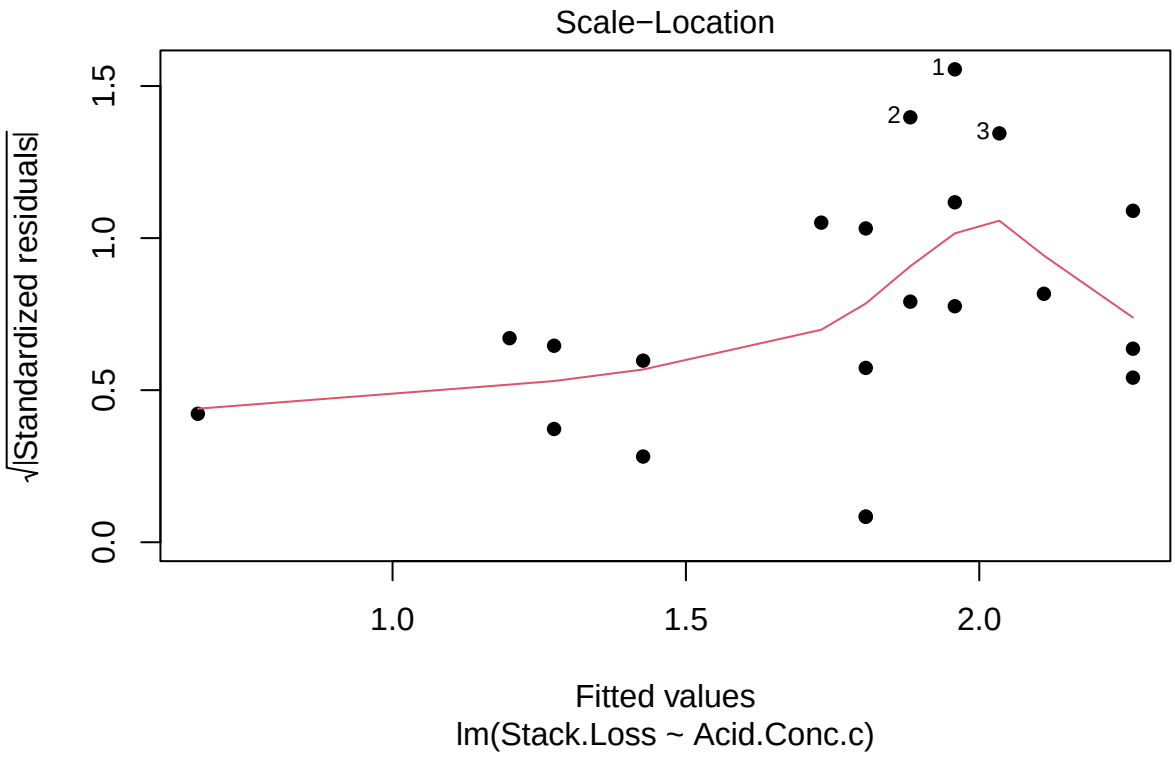

```
plot(Acid.Conc.c, dat$Stack.Loss, ylab = "Stack.Loss")
abline(OLS1)
```



The zero mean model assumption seems to hold. Although for higher fitted values, there is some unevenness: there are more points with negative residuals but a few points with high positive residuals. Overall, the deviance from zero is not too significant. From the second plot we see that there is a somewhat linear trend in which higher values of acid concentration lead to higher values of ammonia loss.

If this assumption was violated, the coefficient estimates would be biased which is problematic as we want $E[\hat{\beta}] = \beta$ so that the model correctly interprets our data.

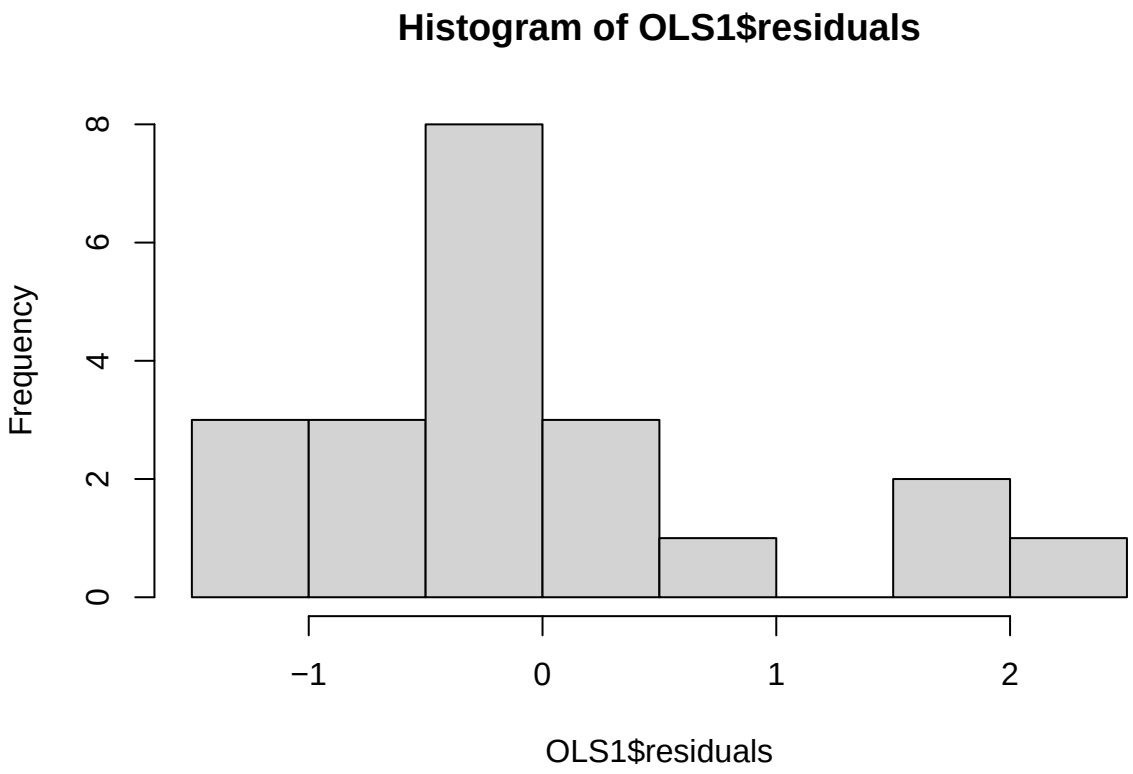
```
# 2. errors have equal variance i.e. homoscedasticity
plot(OLS1, which=3, cex=1, pch=16)  # scale location
```



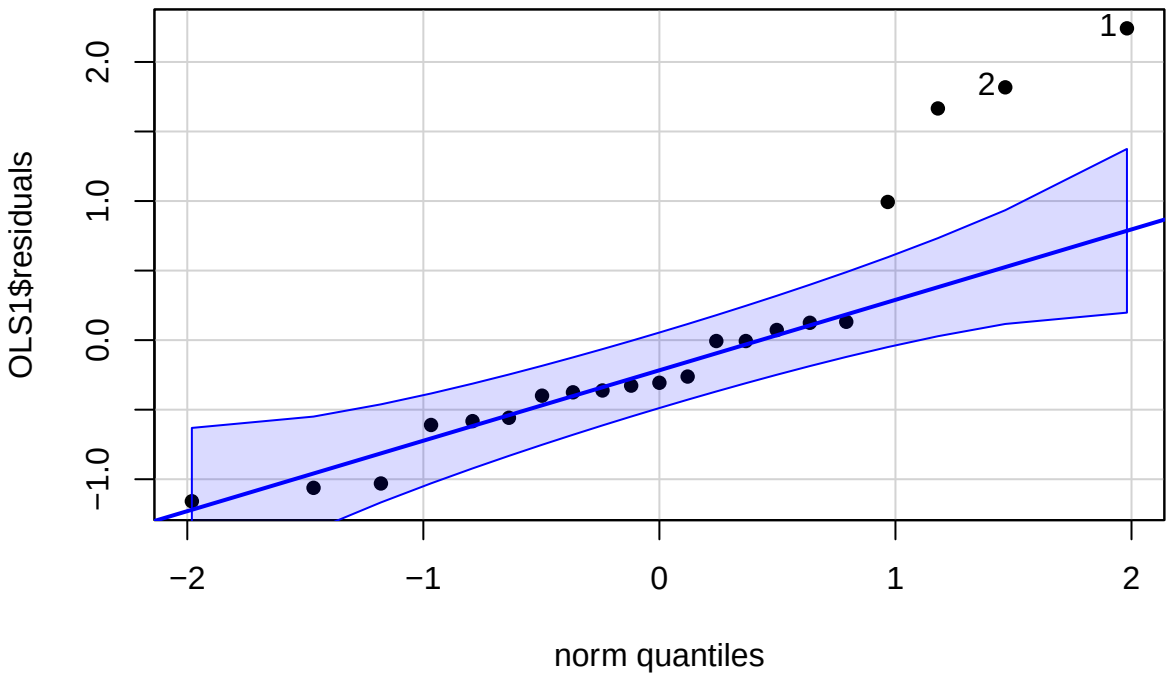
From both the scale-location plot and the residual vs. fitted values plot above, we see that the homoscedasticity assumption is violated. There is a clear fan shape showing that variance increases for higher fitted values. This indicates that the errors are heteroscedastic.

Heteroscedasticity increases the variance of the coefficient estimates and hence it becomes harder to trust the results produced by the linear model summary.

```
# 3. errors are normally distributed
hist(OLS1$residuals)
```




```
qqPlot(OLS1$residuals, cex=1, pch=16)  # q-q plot
```




```
## [1] 1 2
```

The normality assumption is violated. From the histogram we see that the distribution is skewed right. From the q-q plot, we see that there are observations on the right-hand side that do not fall in the confidence band of a normal distribution.

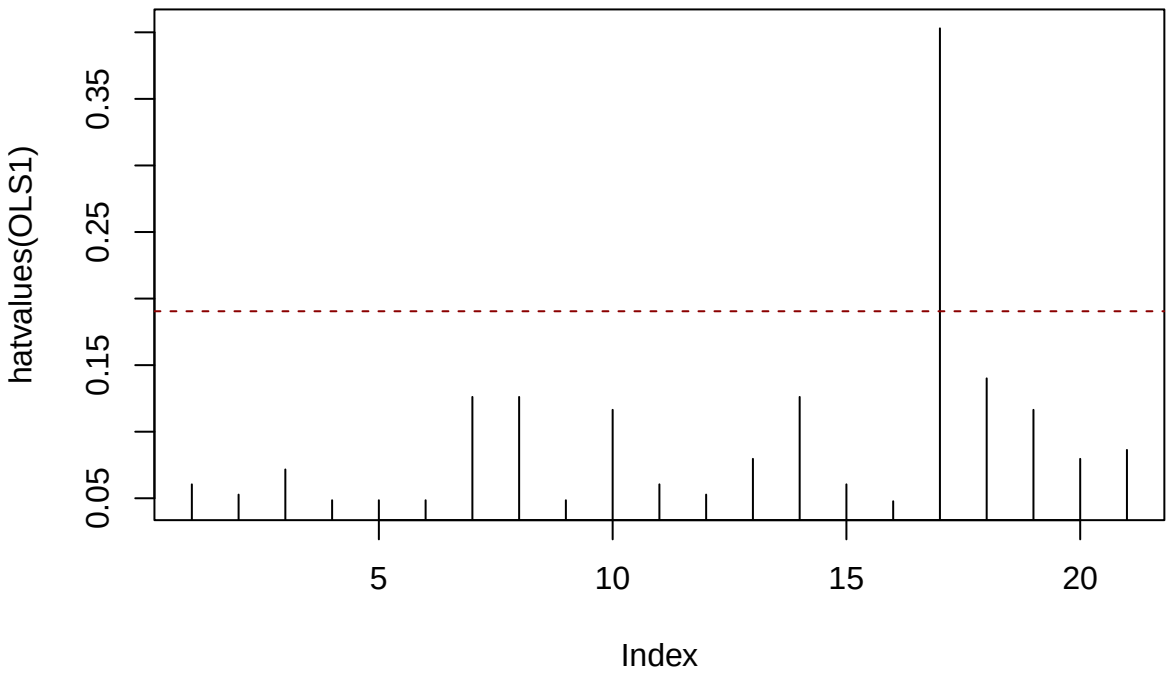
When this assumption is violated, it causes the coefficient estimates to not be normally distributed. This means that we can not trust the associated p-value when deciding whether or not the predictor variable is significant to the model.

It is possible to assess normality of residuals with a histogram and a q-q plot as shown above. However, in our case since we already concluded that the homoscedasticity assumption has been violated, it does not make sense to address the normality assumption since this requires homoscedasticity.

Problem 1D

An outlier in predictor (aka a high-leverage point) is a data point $(\boldsymbol{x_i}, y_i)$ such that $\boldsymbol{x_i}$ is away from the bulk of the sample predictor vectors. We can check this by plotting the hat values:

```
plot(hatvalues(OLS1), type = "h")
p = 1
n = dim(dat)[1]
abline(h = 2 * (p + 1) / n, lty = 2,col = 'darkred')
```



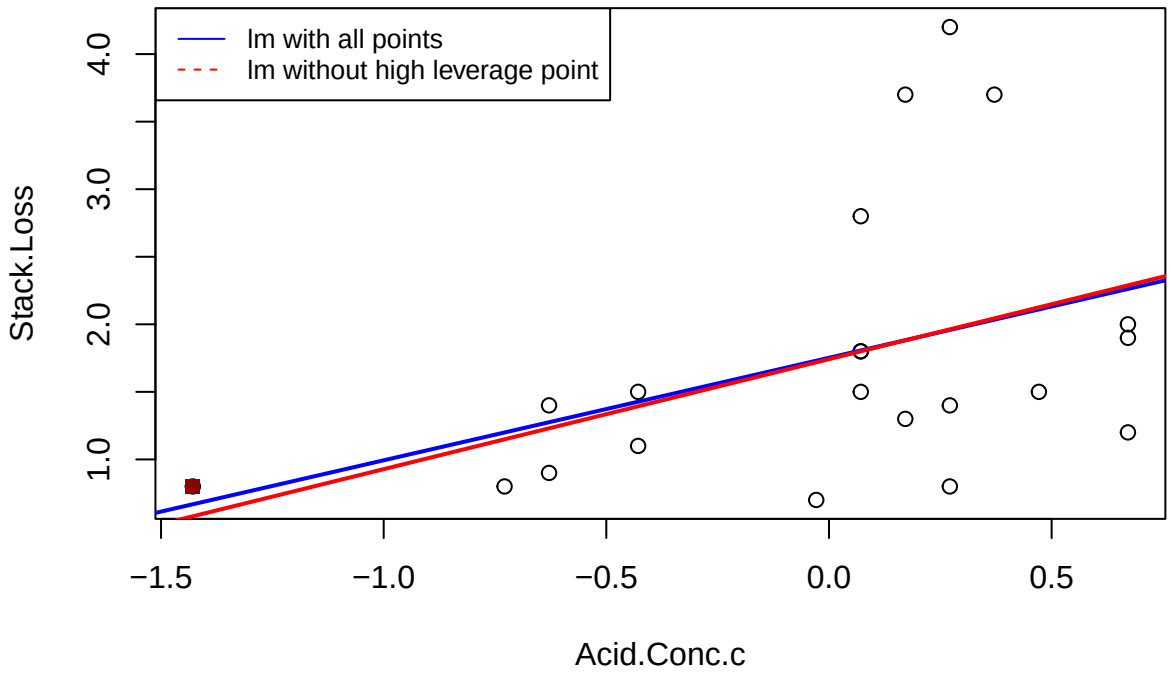
The 17th index has the largest hat value and considered problematic. We can visualize where this observation is compared to the rest and fit a new linear model without this point:

```
plot(Acid.Conc.c, dat$Stack.Loss, ylab = "Stack.Loss")
points(
  Acid.Conc.c[hatvalues(OLS1) > 2 * (p + 1) / n],
  dat$Stack.Loss[hatvalues(OLS1) > 2 * (p + 1) / n],
  col = 'darkred', pch = 15)

# fit lm without high leverage point i.e. 17th index
OLS1.highlev = lm(Stack.Loss[-17] ~ Acid.Conc.c[-17], data = dat)
summary(OLS1.highlev)
```

```
##
## Call:
## lm(formula = Stack.Loss[-17] ~ Acid.Conc.c[-17], data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1628 -0.5674 -0.2965  0.1226  2.2372
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)         1.7419     0.2227   7.823 3.36e-07 ***
## Acid.Conc.c[-17]    0.8139     0.5175   1.573    0.133
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9819 on 18 degrees of freedom
## Multiple R-squared:  0.1208, Adjusted R-squared:  0.07196
## F-statistic: 2.473 on 1 and 18 DF,  p-value: 0.1332
```

```
abline(OLS1, lwd = 2, col = "blue")
abline(OLS1.highlev, lwd = 2, col = "red")
legend("topleft",
       legend=c("lm with all points", "lm without high leverage point"),
       col=c("blue", "red"), lty=1:2, cex=0.8)
```



The highest leverage point is highlighted by the dark red square. It has a significantly lower value of acid concentration than the rest of the observations. However, excluding the highest leverage point does not significantly affect the slope, it only slightly increases it. The slope using all points is 0.7590 while the slope excluding the high-leverage point is 0.8139.

Problem 1E

```
B = 1000  # bootstrap simulations
n = dim(dat)[1]
p = 1
level = 0.95
slopes_boot = rep(NA, B)
```

```r
t1_boot = rep(NA, B)

slope = OLS1$coefficients[2]
se_slope = summary(OLS1)$coefficients[,2][2]

for (i in 1:B) {
  indices = sample(1:n, n, replace=TRUE)
  x_boot = Acid.Conc.c[indices]
  y_boot = dat$Stack.Loss[indices]

  model = lm(y_boot ~ x_boot)
  slopes_boot[i] = model$coefficients[2]
  se_slope_boot = summary(model)$coefficients[,2][2]
  t1_boot[i] = (slopes_boot[i] - slope) / (se_slope_boot)
}

se_boot = sd(slopes_boot)  # bootstrap standard error
ci_boot = matrix(c(slope +
                     quantile(t1_boot, c((1-level)/2,(1+level)/2))*se_slope),
                 ncol = 2)
colnames(ci_boot) = c('2.5 %','97.5 %')  # bootstrap CI

cat('Bootstrap Standard Error for Slope\n')
```

```
## Bootstrap Standard Error for Slope
```

```r
se_boot
```

```
## [1] 0.3016044
```

```r
cat('\n')
```

```r
cat('OLS Standard Error for Slope\n')
```

```
## OLS Standard Error for Slope
```

```r
se_slope
```

```
## Acid.Conc.c
##   0.3991529
```

```r
cat('\n')
```

```r
cat('Bootstrap CI for Slope\n')
```

```
## Bootstrap CI for Slope
```

```
ci_boot
```

```
##           2.5 %   97.5 %
## [1,] 0.09518051 1.307806
```

```
cat('\n')
```

```
cat('OLS CI for Slope\n')
```

```
## OLS CI for Slope
```

```
confint(OLS1)[2,]
```

```
##       2.5 %      97.5 %
## -0.07648142  1.59439187
```

The bootstrap standard error for slope is lower than the OLS one. The bootstrap CI for slope is narrower than the OLS one.

Bootstrapping performs better with larger sample sizes since with more samples, we learn more about the original distribution. Bootstrapping also does not make any assumptions about the underlying distribution. Previously, we saw that the OLS assumptions were violated. Hence, we can not trust the standard OLS estimates of standard error and confidence interval for the slope parameter. We expect the bootstrap estimates to be better. However, we must note that within the bootstrap simulations we are fitting a linear model by least squares which also requires the OLS assumptions. Hence, the bootstrap estimates are also not accurate.

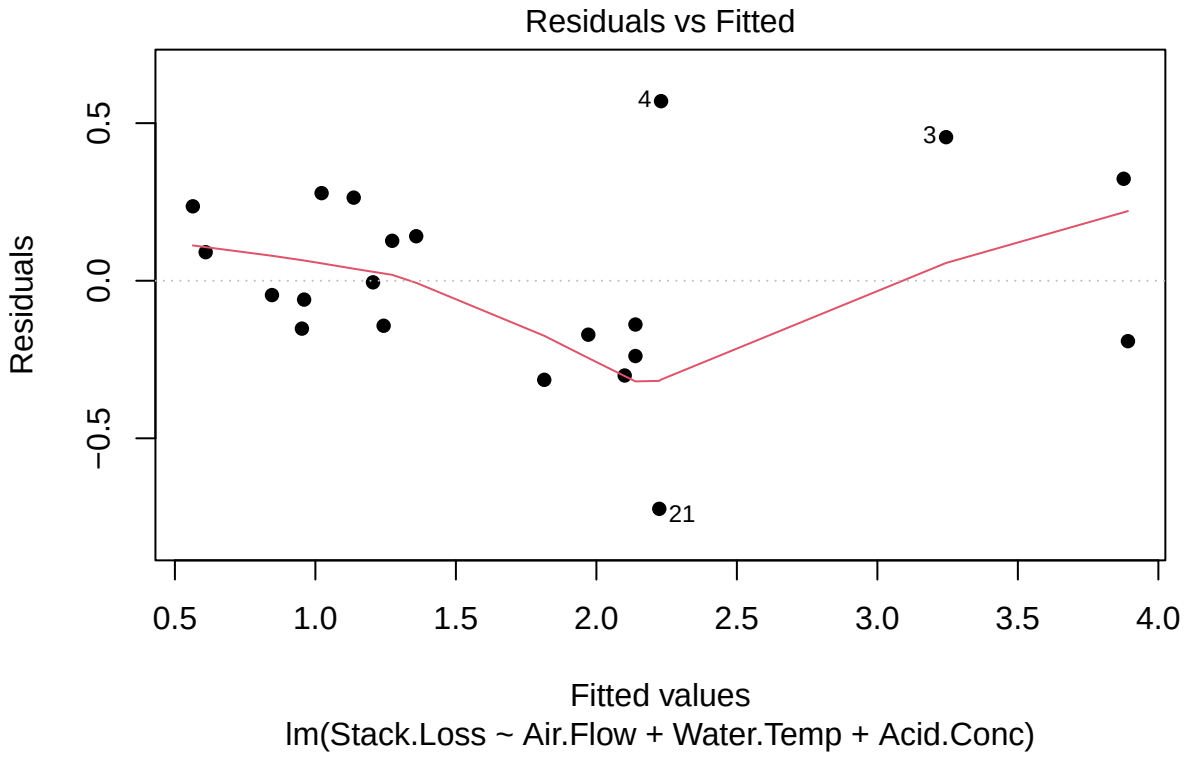
Problem 2A

```
OLS.mult = lm(Stack.Loss ~ Air.Flow + Water.Temp + Acid.Conc, data = dat)
summary(OLS.mult)
```

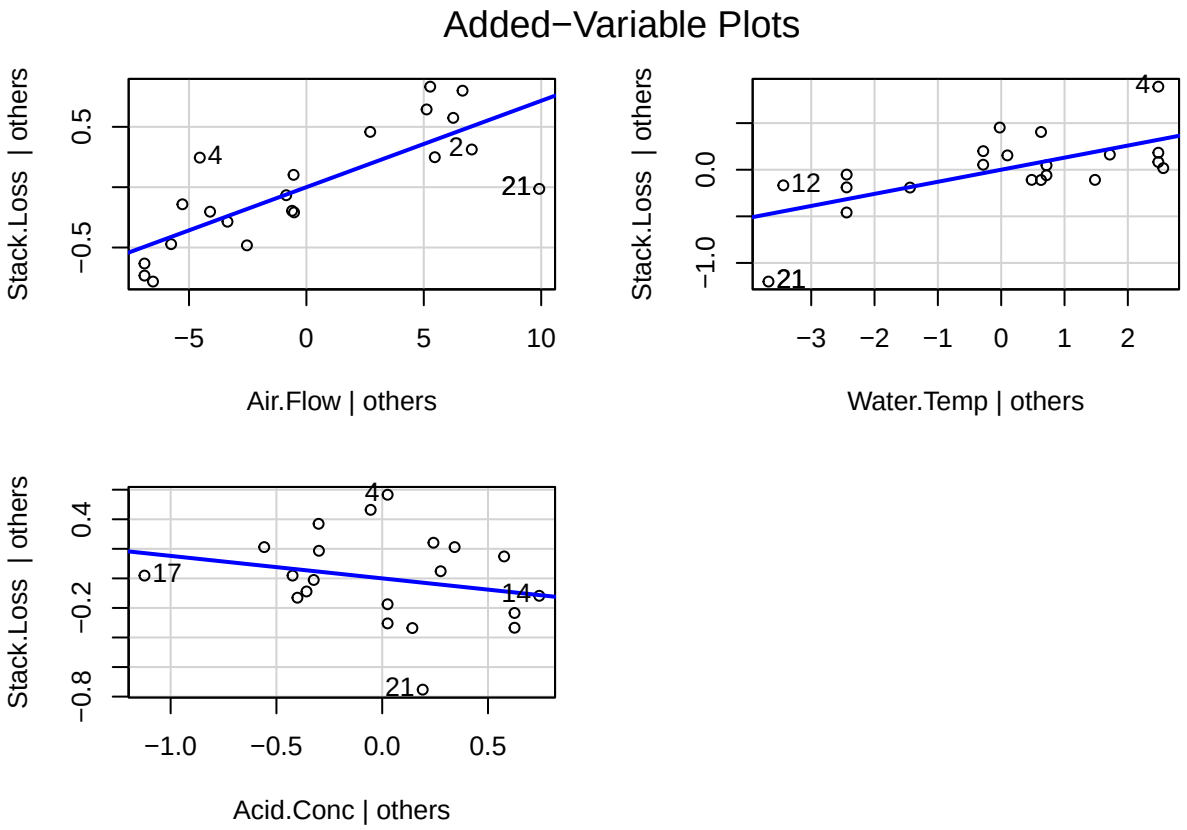
```
##
## Call:
## lm(formula = Stack.Loss ~ Air.Flow + Water.Temp + Acid.Conc,
##     data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.72377 -0.17117 -0.04551  0.23614  0.56978
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.61416    8.90213   0.406  0.68982
## Air.Flow     0.07156    0.01349   5.307  5.8e-05 ***
## Water.Temp   0.12953    0.03680   3.520  0.00263 **
## Acid.Conc   -0.15212    0.15629  -0.973  0.34405
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.3243 on 17 degrees of freedom
## Multiple R-squared:  0.9136, Adjusted R-squared:  0.8983
## F-statistic:  59.9 on 3 and 17 DF,  p-value: 3.016e-09
```

```
# 1. errors have zero mean
plot(OLS.mult, which=1, pch=16)  # residuals vs. fitted values
```

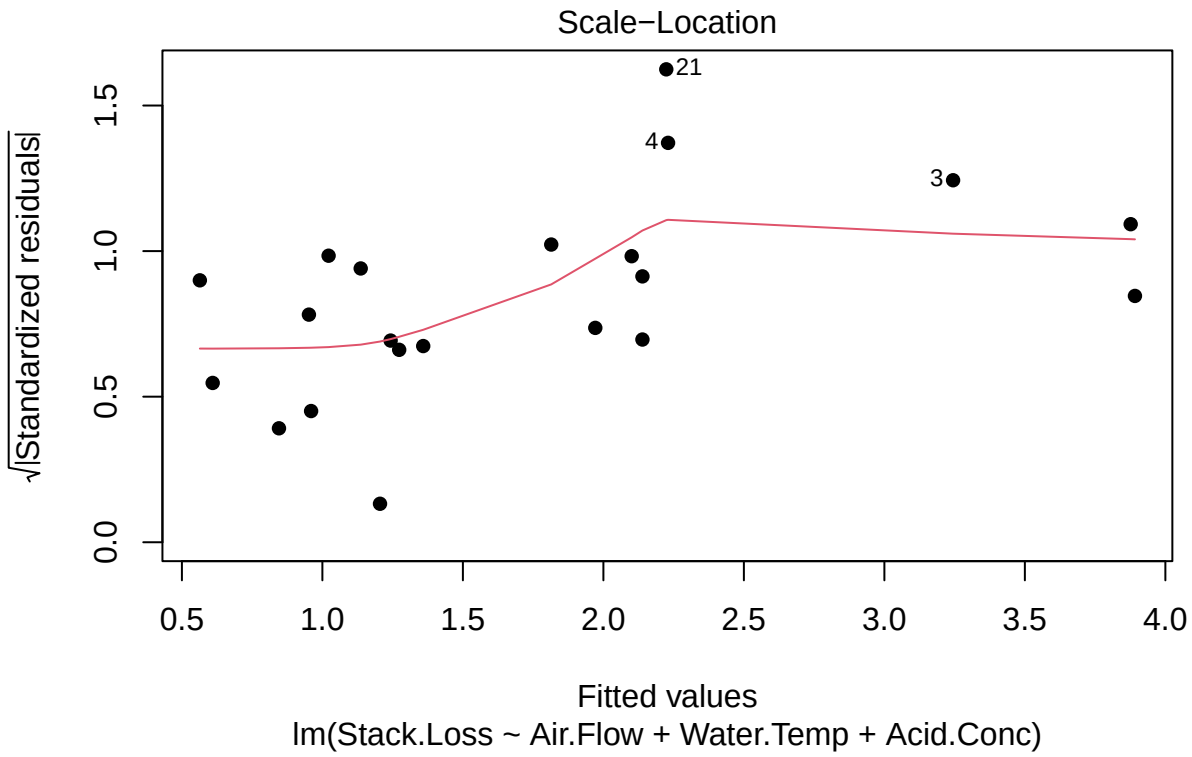



```
avPlots(OLS.mult)  # added variable: regressing out other variables
```

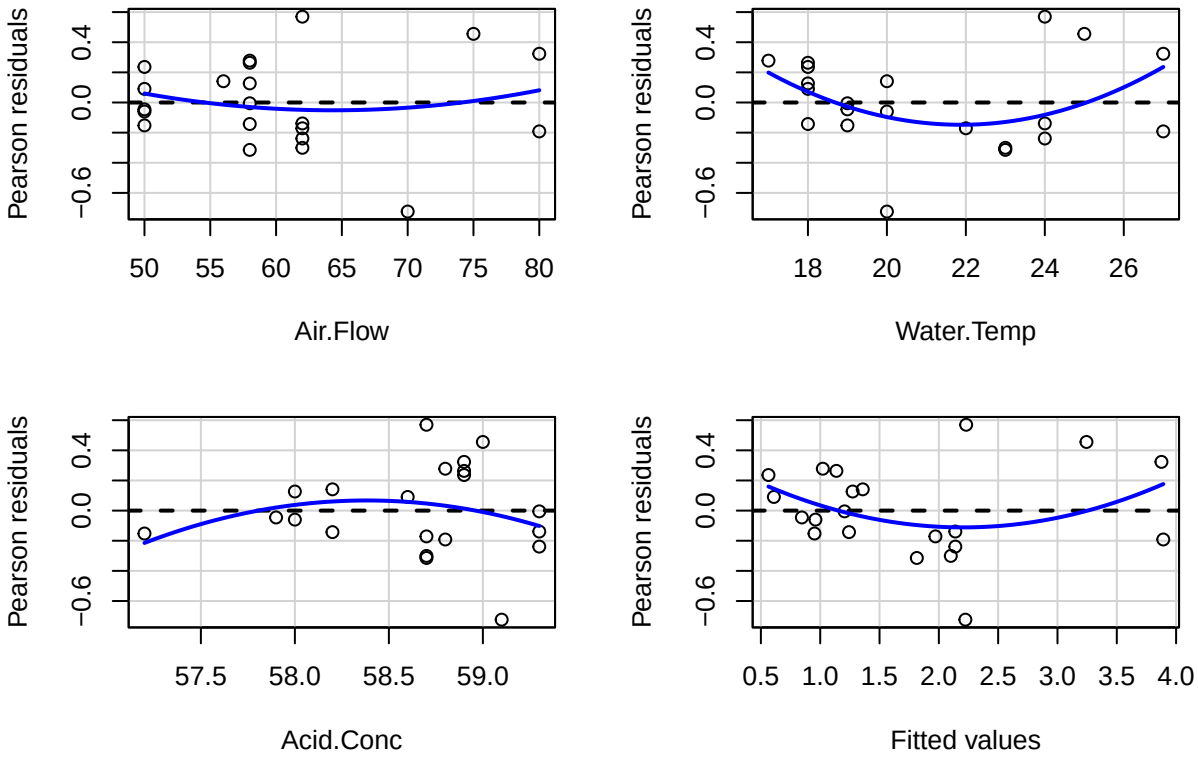


The mean zero assumption is violated. The residuals vs. fitted values plot is clearly not centered around zero nor scattered evenly. We see a quadratic shape. This indicates that the errors do not have mean zero. This differs from Problem 1C as we only observed a slight deviance from the horizontal line at 0. The added variable plots show that Air.Flow and Water.Temp seem to have good linear fits (except for a few outliers), but Acid.Conc is a bit spread out from the line.

```
# 2. errors have equal variance i.e. homoscedasticity
plot(OLS.mult, which=3, cex=1, pch=16)  # scale location
```



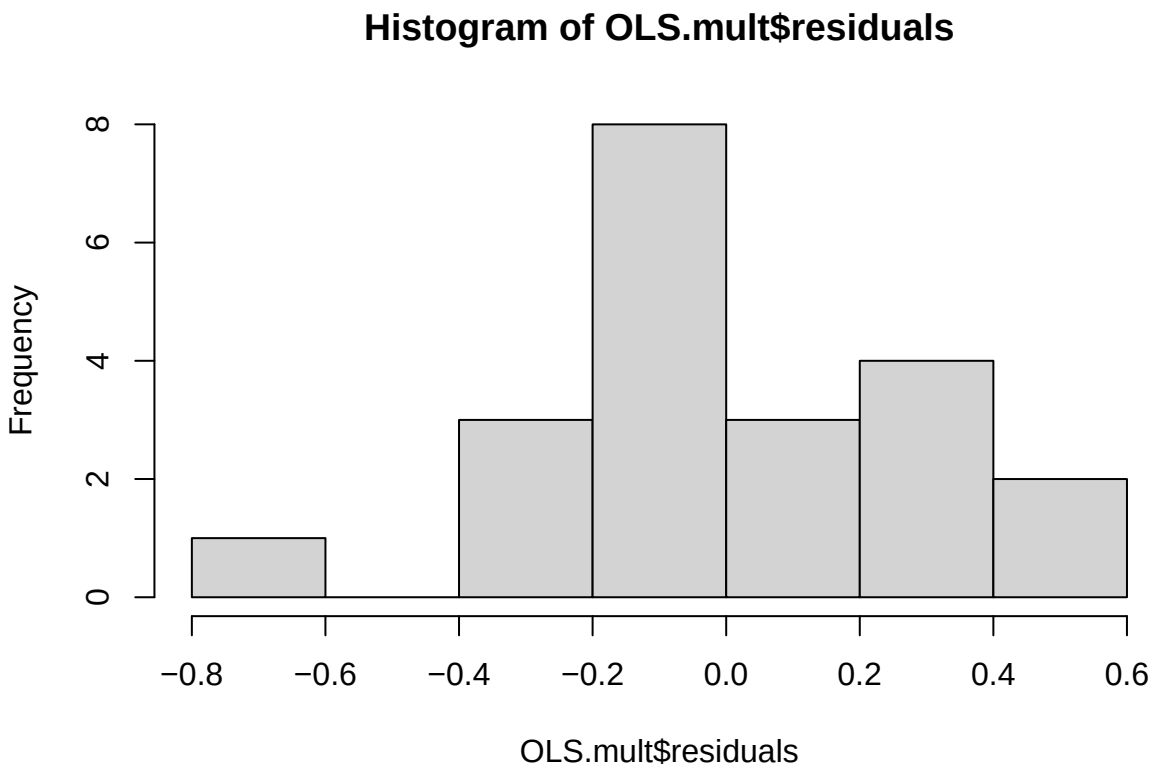

```
residualPlots(OLS.mult)  # partial residual: looking at one variable
```



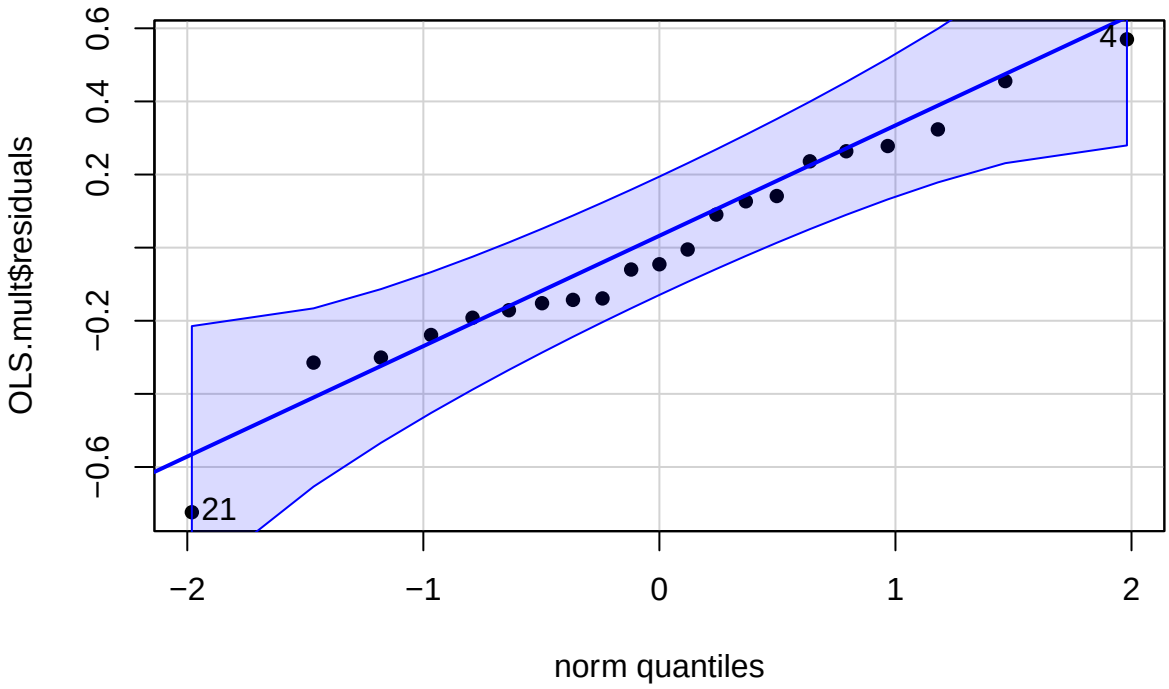

```
##             Test stat Pr(>|Test stat|)
## Air.Flow       0.7594          0.45866
## Water.Temp     2.1794          0.04459 *
## Acid.Conc     -1.0038          0.33040
## Tukey test     1.5033          0.13277
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The homoscedasticity assumption is violated. We see a fan shape in the residuals vs. fitted values plot. When looking at the variables separately, we particularly see a fan shape for Air.Flow and Acid.Conc. For Water.Temp, we see a curvature indicating the errors are not evenly scattered around mean zero. Also, in the scale-location plot the standardized residuals do not form a horizontal line. This is similar to Problem 1C as we also found that the assumption was violated there.

```
# 3. errors are normally distributed
hist(OLS.mult$residuals)
```




```
qqPlot(OLS.mult$residuals, cex=1, pch=16)  # q-q plot
```




```
## [1] 21  4
```

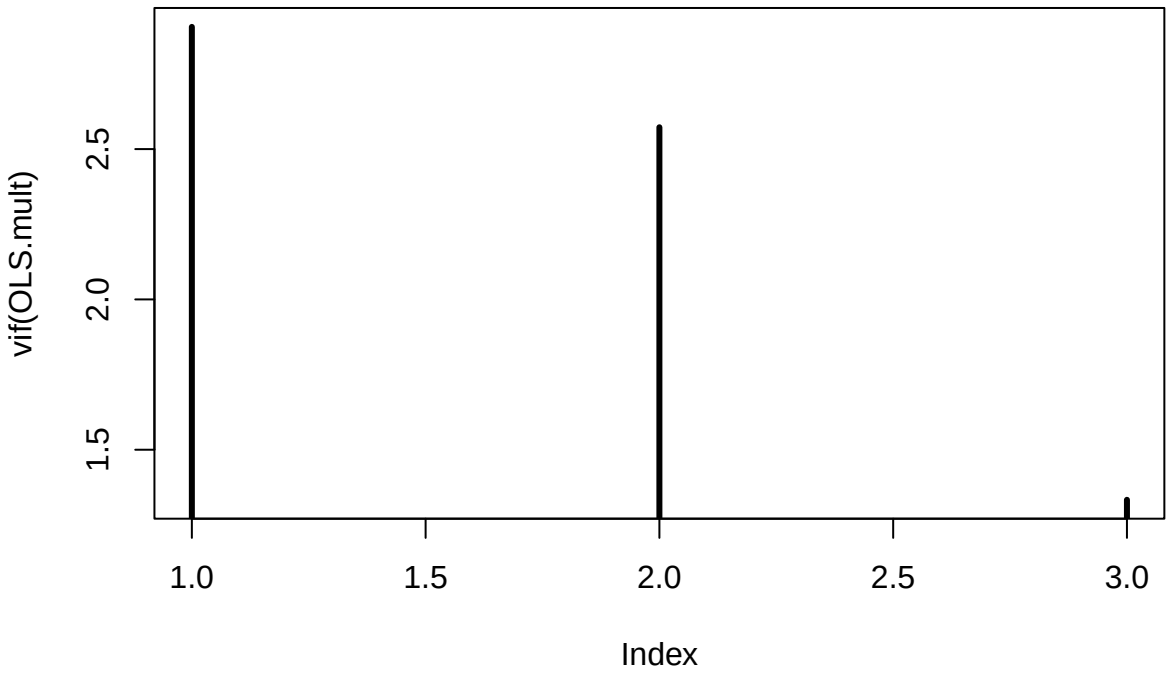
The histogram contains a gap at the lower end of the residuals but is otherwise not too far off from being normal. The q-q plot shows that all the points lie within the confidence band indicating that the errors are normally distributed. This differs from Problem 1C as there were several points that did not fit into the band and hence we concluded that the assumption was violated. However, since we already concluded for this case that the mean zero and homoscedasticity assumptions were violated, it follows that the errors are not normally distributed with mean zero and equal variance.

Problem 2B

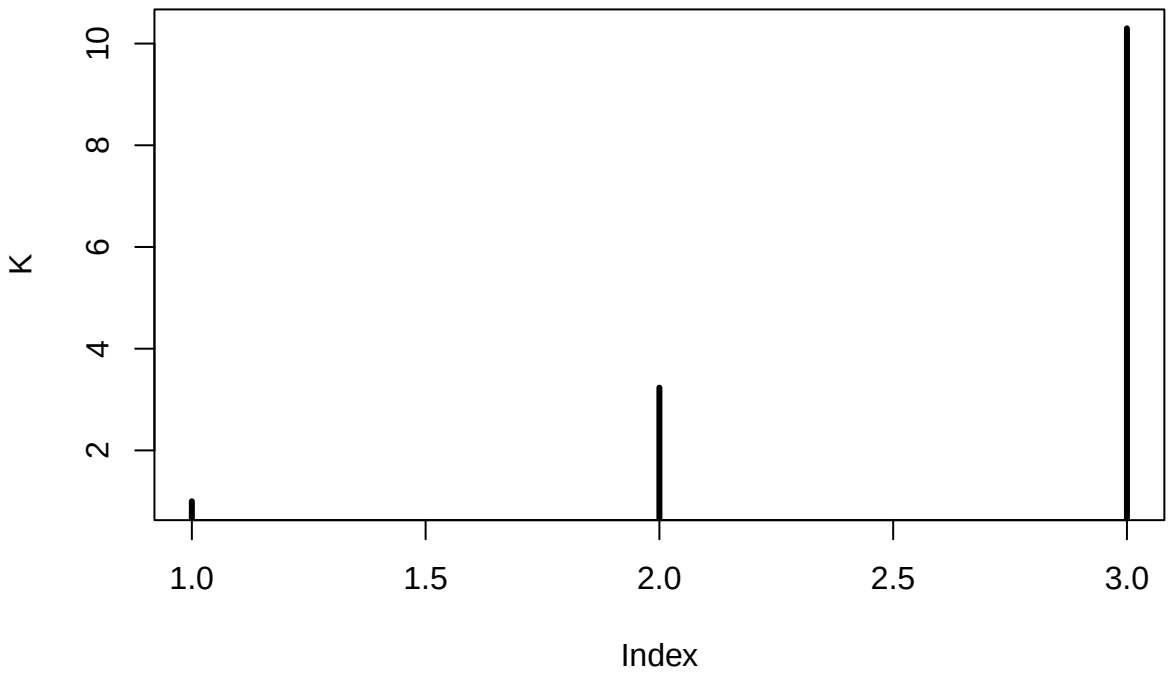
```
# multicollinearity

# checking via pairwise correlations b/w predictors
dat_X = scale(dat[-4])
round( cor(dat_X) , 2) # rounded to 2 digits
```

```
##             Air.Flow Water.Temp Acid.Conc
## Air.Flow        1.00       0.78      0.50
## Water.Temp      0.78       1.00      0.39
## Acid.Conc       0.50       0.39      1.00
```

```r
# checking via variance inflation factors (VIF)
plot(vif(OLS.mult), type='h', lwd=3)
abline(h = 10, lty=2) # threshold for suspects
```




```r
# checking via condition indices
C = cor(dat_X) # correlation matrix for the predictors
L = eigen(C) # eigenvalues
K = max(L$val)/L$val # condition indices
plot(K, type='h', lwd=3)
abline(h = 1000, lty=2) # threshold for suspects
```



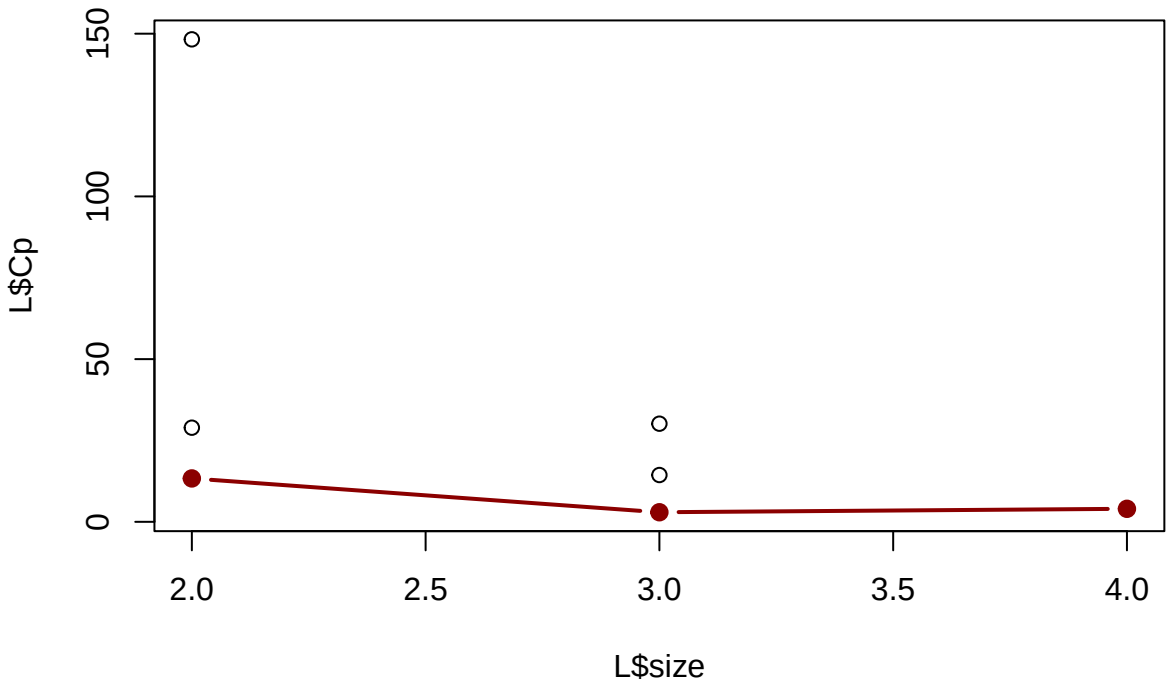
Looks like multicollinearity is not an issue. None of the correlations in the map have absolute value above 0.8 (although Air.Flow and Water.Temp are on the border of being concerned about). In addition, for each variable the VIF stays below 10 and the condition number below 1000.

We must check for multicollinearity because serveral issues arise if the linear predictors are nearly dependent:

- interpretation is difficult
- the coefficient estimates have large variances
- the fit is numerically unstable since $X^T X$ is almost singular

Problem 2C

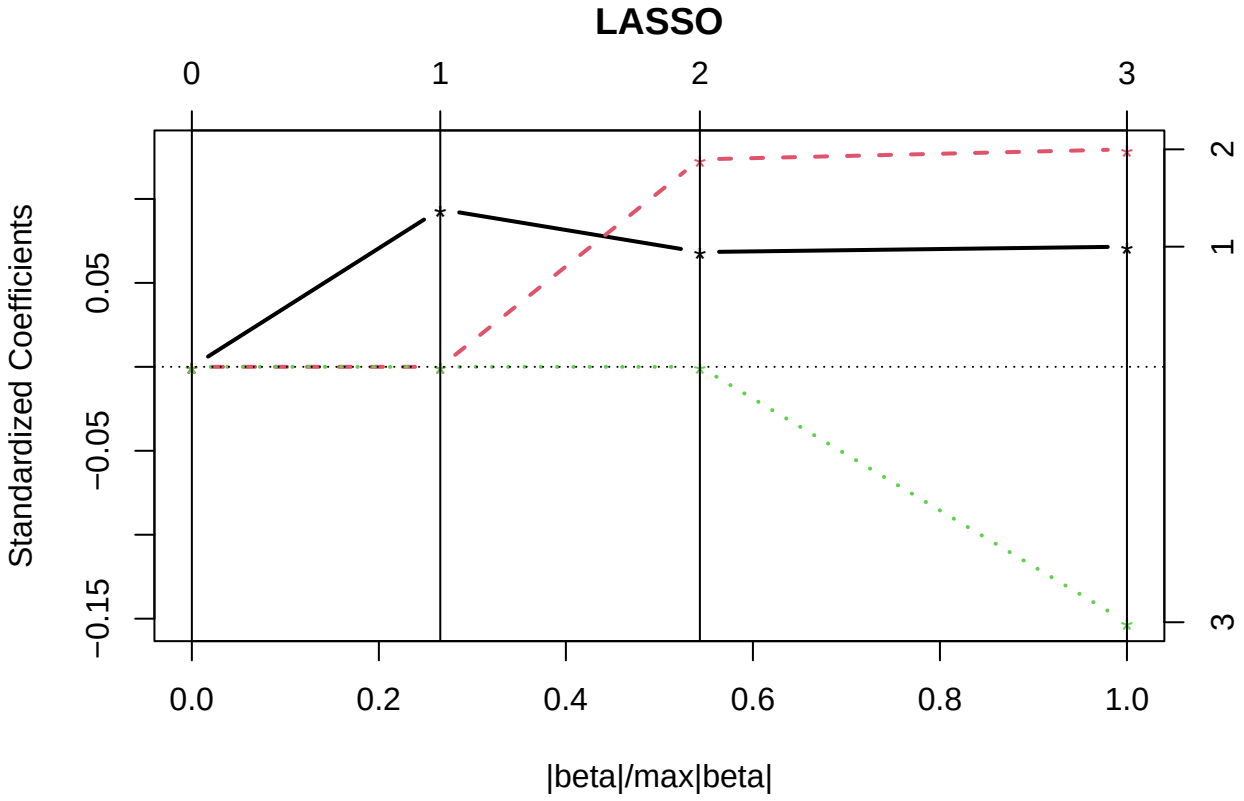
```
# best subset regression model via Cp
L = leaps(dat[-4], dat[,4])
ind = which.min(L$Cp)
plot(L$size,L$Cp)
points(L$size[ind],L$Cp[ind],col = 'darkred',pch = 19)
points(aggregate(L$Cp, by = list(L$size), min),lwd = 2,col = 'darkred',type = 'b',pch = 19)
```




```
# best model selected
names(dat[-4])[L$which[ind,]]
```

```
## [1] "Air.Flow"   "Water.Temp"
```

```
# best subset via LASSO
lasso.fit = lars(x = model.matrix(OLS.mult)[,-1], y = dat[,4], normalize=FALSE, intercept=TRUE)
plot(lasso.fit, lwd=2)
```




```
# best model selected
lasso.fit$actions
```

```
## [[1]]
## Air.Flow
##        1
##
## [[2]]
## Water.Temp
##          2
##
## [[3]]
## Acid.Conc
##         3
```

```
lasso.fit$Cp
```

```
##          0          1          2          3
## 177.706678  14.408610   2.976081   4.000000
## attr(,"sigma2")
##         3
## 0.1051941
## attr(,"n")
## [1] 21
```

```
# With 2 variables, we get the lowest Cp at 2.98, meaning the best model selected is Air.Flow + Water.T
```

Both methods choose the model $Stack.Loss \sim Air.Flow + Water.Temp$. This was expected because when we ran the summary of the OLS, Acid.Conc did not have a low enough p-value indicating that the variable is not significant to the model. Also, when looking at the added variable plots, Air.Flow and Water.Temp seemed to have a stronger linear relationship to Stack.Loss, so it would be sufficient to keep these predictors and drop Acid.Conc.

Problem 3A

```
BackwardPath <- function(x, y) {

  n = dim(x)[1]  # num of observations
  k = dim(x)[2]  # num of predictors
  all_best_AICs = rep(NA, k+1)
  names(all_best_AICs) = k:0
  all_best_coefs = list()

  # full model
  fit = lm(y ~ ., data = x)
  y_pred = predict(fit)
  all_best_coefs[1] = list(fit$coefficients)
  RSS = sum((y - y_pred) ^ 2)
  all_best_AICs[1] = n * log(RSS / n) + 2 * k

  for(i in k:1){

    # fit all models with i - 1 predictors
    candidates = combn(1:k, i - 1, simplify = FALSE)
    candidate_AICs = rep(NA, length(candidates))
    candidate_coefs = list()

    for(j in 1:length(candidates)) {

      if(i - 1 == 0) {
        fit = lm(y ~ 1)
      } else {
        current_features = unlist(candidates[j])
        fit = lm(y ~ ., data = data.frame(x[current_features]))
      }
      y_pred = predict(fit)
      candidate_coefs[j] = list(fit$coefficients)

      RSS = sum((y - y_pred) ^ 2)
      candidate_AICs[j] = n * log(RSS / n) + 2 * (i - 1)
    }

    all_best_AICs[k-i+2] = min(candidate_AICs)
    all_best_coefs[k-i+2] = candidate_coefs[which.min(candidate_AICs)]
  }
```

```
  return(list(all_best_AICs = all_best_AICs, all_best_coefs = all_best_coefs))
}
```

```
# testing
x = dat[, -which(names(dat) %in% c("Stack.Loss"))]
y = dat$Stack.Loss
coeffPath = BackwardPath(x, y)
coeffPath$all_best_AICs
```

```
##           3           2           1           0
## -45.7284013 -46.5896108 -37.5668900  -0.3098944
```

```
coeffPath$all_best_coefs
```

```
## [[1]]
## (Intercept)    Air.Flow  Water.Temp   Acid.Conc
##  3.61415852  0.07156402  0.12952861 -0.15212252
##
## [[2]]
## (Intercept)    Air.Flow  Water.Temp
## -5.03588401  0.06711544  0.12953514
##
## [[3]]
## (Intercept)    Air.Flow
##  -4.4132025   0.1020309
##
## [[4]]
## (Intercept)
##    1.752381
```

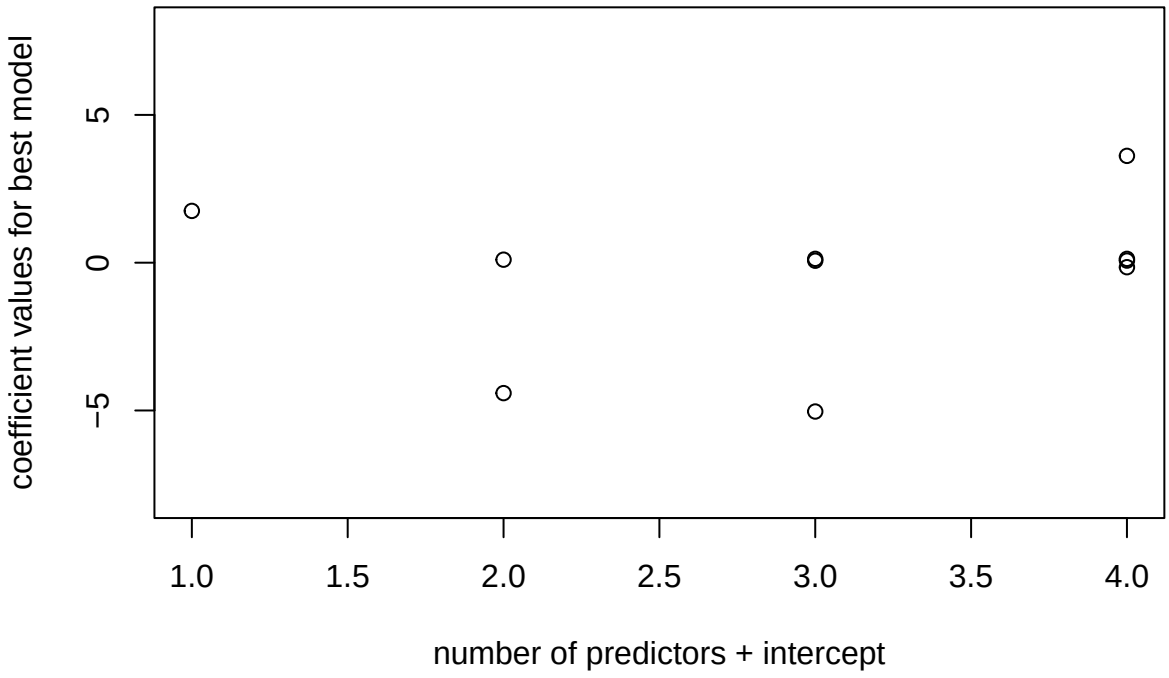
Problem 3B

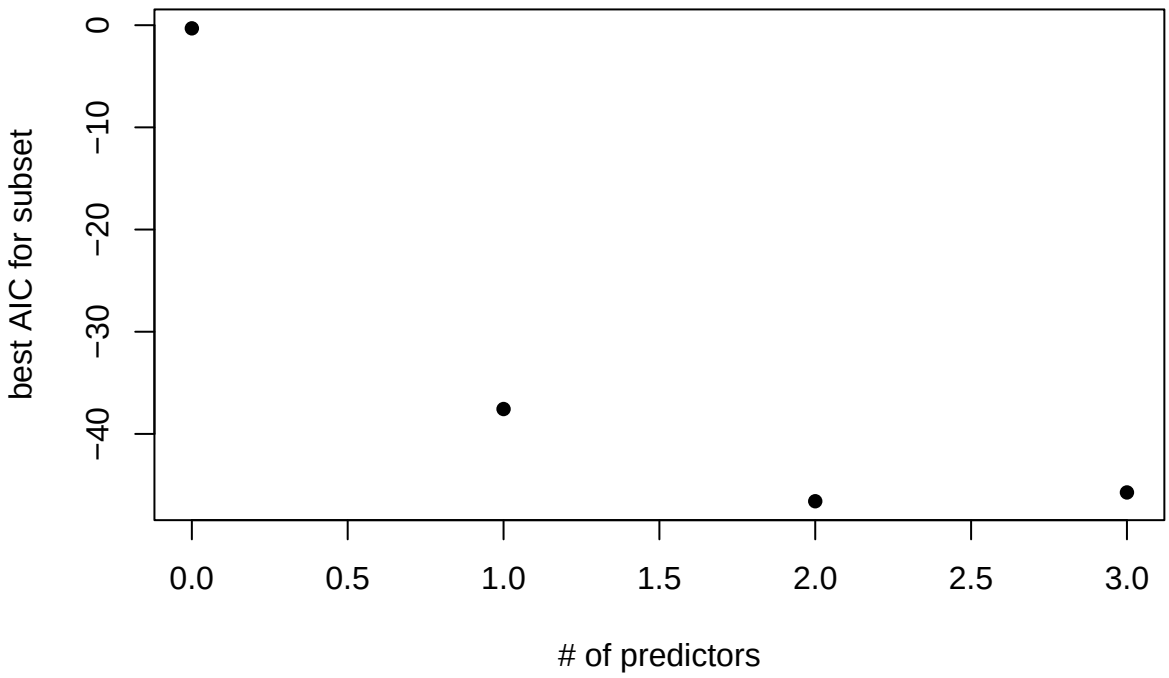
```
BackwardVisualize <- function(coeffPath) {
  AICs = coeffPath$all_best_AICs
  coefs = coeffPath$all_best_coefs
  predictors = (length(coefs) - 1):0

  # plot coefs for each of the predictors vs. # of predictors
  for(i in 1:length(coeffPath$all_best_coefs)) {
    num_coefs = length(coeffPath$all_best_coefs[[i]])
    if(i==1){
      plot(rep(num_coefs, num_coefs),
           coeffPath$all_best_coefs[[i]],
           xlab="number of predictors + intercept",
           ylab="coefficient values for best model",
           xlim=c(1,num_coefs),
           ylim=c(-8,8))
    } else {
      points(rep(num_coefs, num_coefs),
             coeffPath$all_best_coefs[[i]])
```

```
    }
  }

  # plot AIC vs. # of predictors
  plot(predictors, AICs,
       xlab="# of predictors", ylab="best AIC for subset", pch=16)
}
```

Problem 3C

```
x = dat[, -which(names(dat) %in% c("Stack.Loss"))]
y = dat$Stack.Loss
coeffPath = BackwardPath(x, y)
BackwardVisualize(coeffPath)
```





The best subset regression model according to this algorithm is $Stack.Loss \sim Air.Flow + Water.Temp$ because the AIC value is lowest with 2 predictors and we can see which 2 predictors those are via the coefficient list variable. This is exactly the same result that we saw in Problem 2C.

Problem 4A

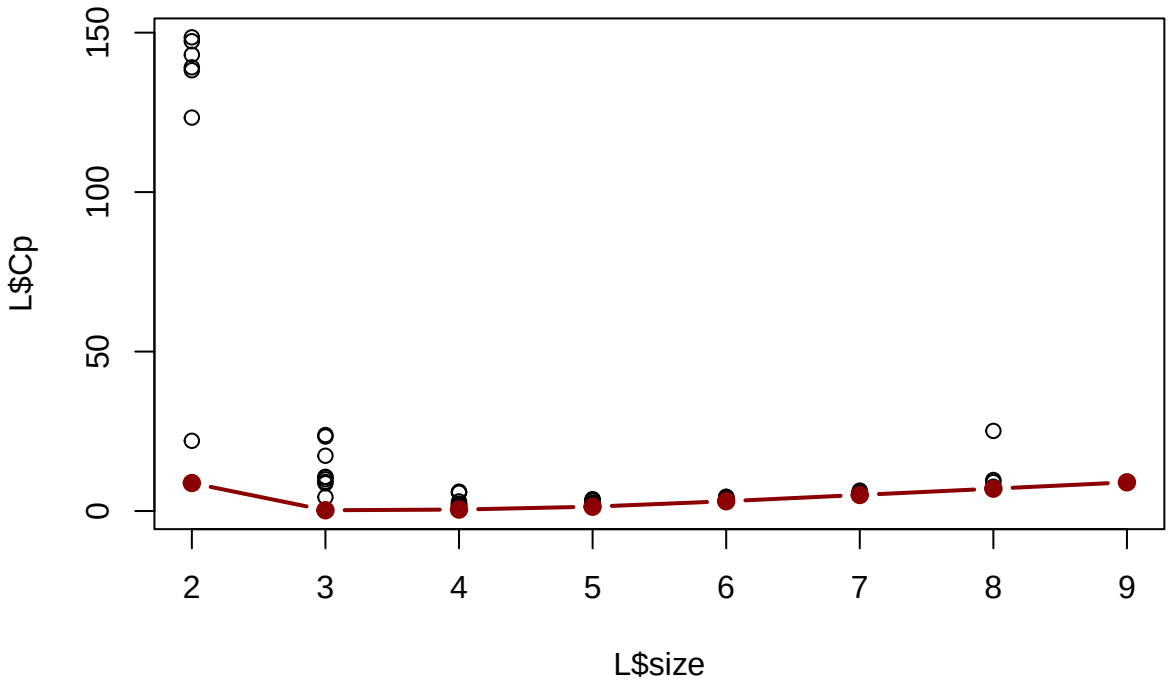
```
n = nrow(dat)
x = matrix(runif(n*5, -1, 1), nrow=n)
colnames(x) = c("x1", "x2", "x3", "x4", "x5")
dat_junk = cbind(dat, x)

fit = lm(Stack.Loss ~ ., data = dat_junk)
summary(fit)
```

```
##
## Call:
## lm(formula = Stack.Loss ~ ., data = dat_junk)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.60834 -0.18756 -0.01588  0.10418  0.46799
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)  1.89593   11.47464   0.165  0.87152
## Air.Flow     0.07944    0.01866   4.257  0.00111 **
## Water.Temp   0.09439    0.05717   1.651  0.12465
## Acid.Conc   -0.11730    0.20161  -0.582  0.57146
## x1          -0.12637    0.16099  -0.785  0.44769
## x2          -0.23570    0.16403  -1.437  0.17629
## x3          -0.02586    0.22017  -0.117  0.90845
## x4           0.03490    0.14845   0.235  0.81811
## x5           0.01478    0.17764   0.083  0.93508
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3519 on 12 degrees of freedom
## Multiple R-squared:  0.9282, Adjusted R-squared:  0.8803
## F-statistic: 19.38 on 8 and 12 DF,  p-value: 9.533e-06
```

Problem 4B

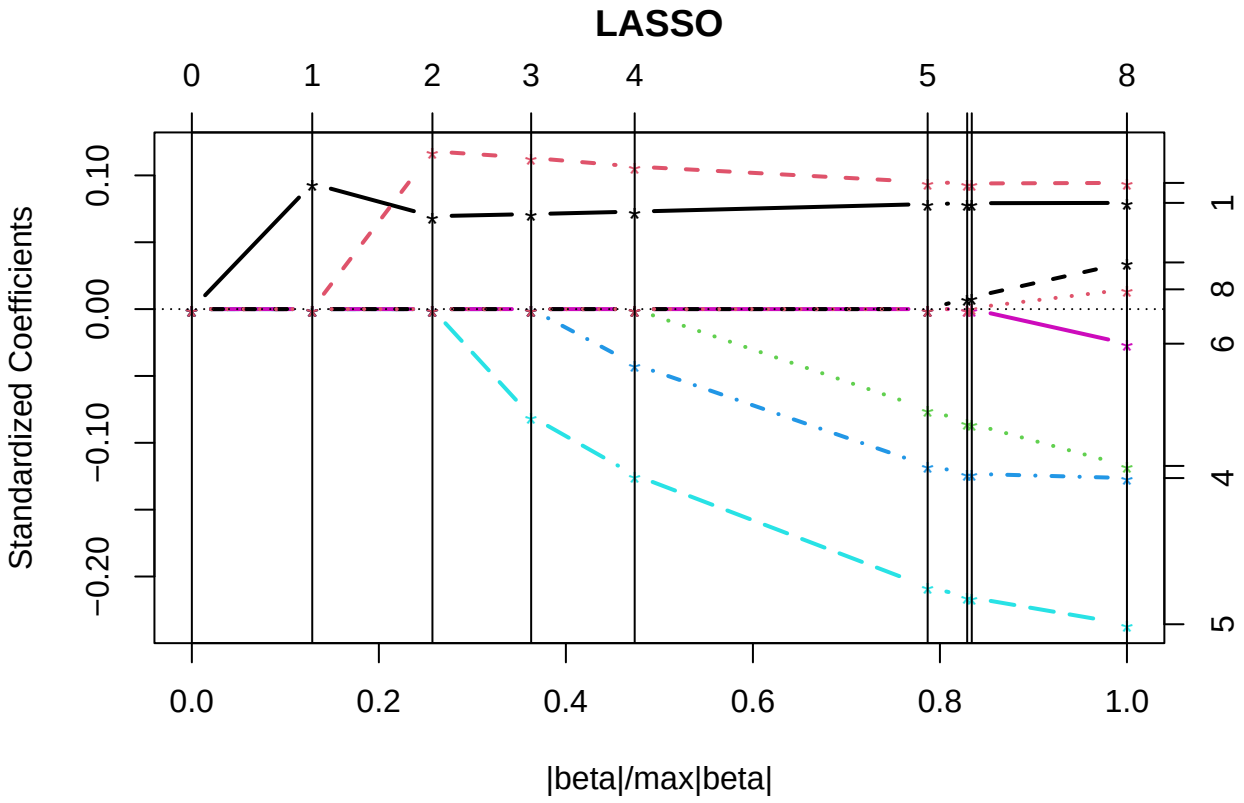
```
# repeat problem 2C
X = dat_junk[, -which(names(dat_junk) %in% c("Stack.Loss"))]
y = dat_junk$Stack.Loss

# best subset regression model via Cp
L = leaps(X, y)
ind = which.min(L$Cp)
plot(L$size,L$Cp)
points(L$size[ind],L$Cp[ind],col = 'darkred',pch = 19)
points(aggregate(L$Cp, by = list(L$size), min),lwd = 2,col = 'darkred',type = 'b',pch = 19)
```




```
# best model selected
names(X)[L$which[ind,]]
```

```
## [1] "Air.Flow"   "Water.Temp"
```

```
# repeat problem 2C

# best subset via LASSO
lasso.fit = lars(x = model.matrix(fit)[,-1], y = y, normalize=FALSE, intercept=TRUE)
plot(lasso.fit, lwd=2)
```




```
# best model selected
lasso.fit$actions
```

```
## [[1]]
## Air.Flow
##        1
##
## [[2]]
## Water.Temp
##          2
##
## [[3]]
## x2
##  5
##
## [[4]]
## x1
##  4
##
## [[5]]
## Acid.Conc
##         3
##
## [[6]]
## x4
##  7
```

```
##
## [[7]]
## x5
##  8
##
## [[8]]
## x3
##  6
```

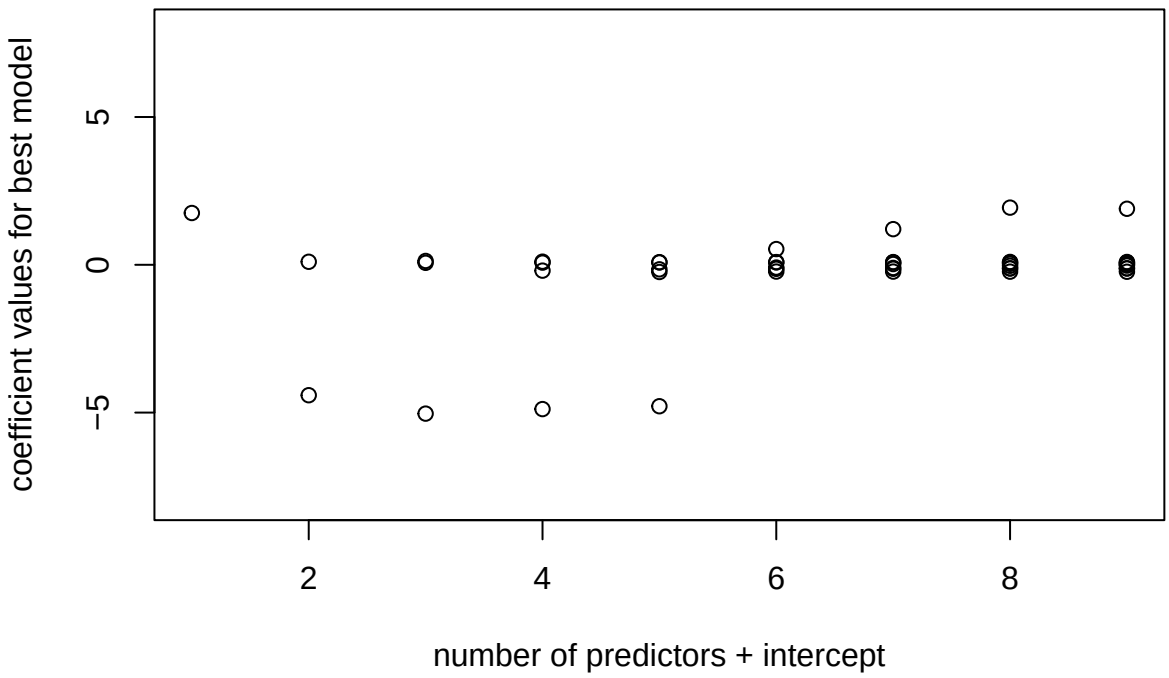
```
lasso.fit$Cp
```

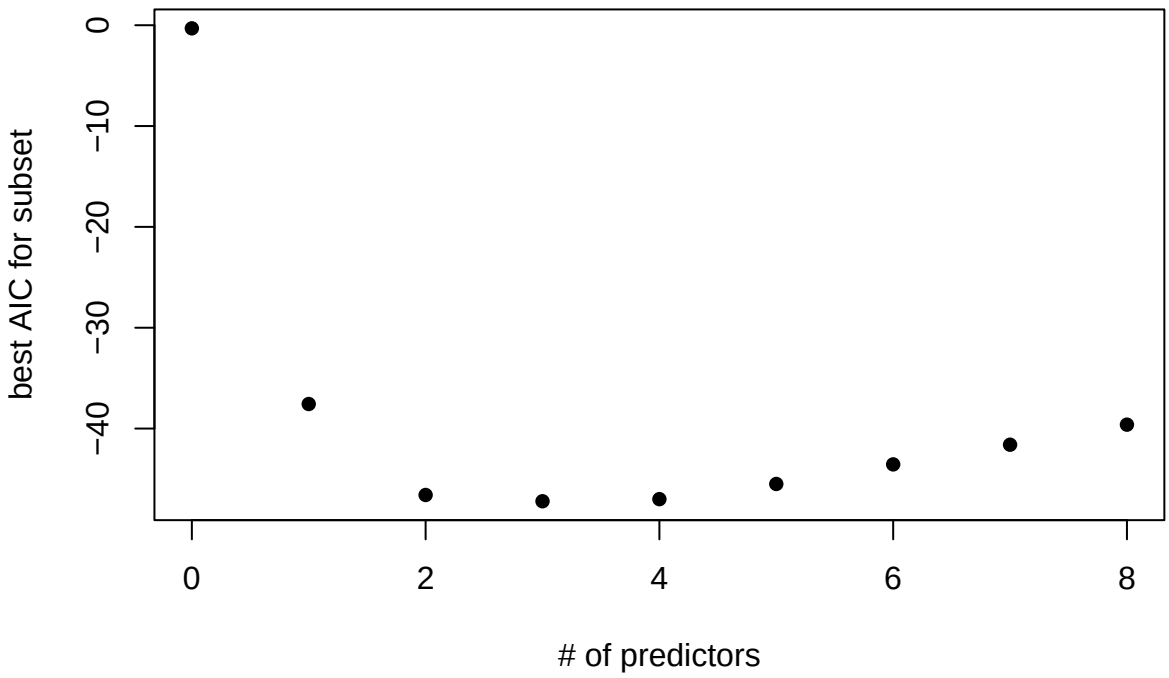
```
##           0           1           2           3           4           5
## 148.0765265   9.6774949   0.3346947   1.1193544   2.2971653   3.1171959
##           6           7           8
##   5.0724608   7.0683513   9.0000000
## attr(,"sigma2")
##         8
## 0.1238497
## attr(,"n")
## [1] 21
```

```
# model with 2 variables has min Cp, hence we look at the first 2 actions which are Air.Flow and Water.
```

Repeating Problem 2C via Cp on subset regression models and on subset LASSO fits, we get the same variable selection: Air.Flow + Water.Temp. Hence, the junk variables do not mess up the variable selection.

```
# repeat problem 3C

coeffPath = BackwardPath(X, y)
BackwardVisualize(coeffPath)
```


coefficient values for best model
5
0
−5
2
4
6
8
number of predictors + intercept




```
min(coeffPath$all_best_AICs)
```

```
## [1] -47.2091
```

```
coeffPath$all_best_coefs[which.min(coeffPath$all_best_AICs)]
```

```
## [[1]]
## (Intercept)    Air.Flow  Water.Temp          x2
## -4.88229512  0.07333482  0.10599446 -0.19771637
```

Repeating Problem 3C via the BackwardPath algorithm, we get a different variable selection: Air.Flow + Water.Temp + Acid.Conc + x2 + x3. Hence, the junk variables mess up the variable selection. Compared to 2C, we now have three more variables in the model, two of which are junk and one of which was excluded after looking at subsets and concluding that Acid.Conc does not help the model prediction.

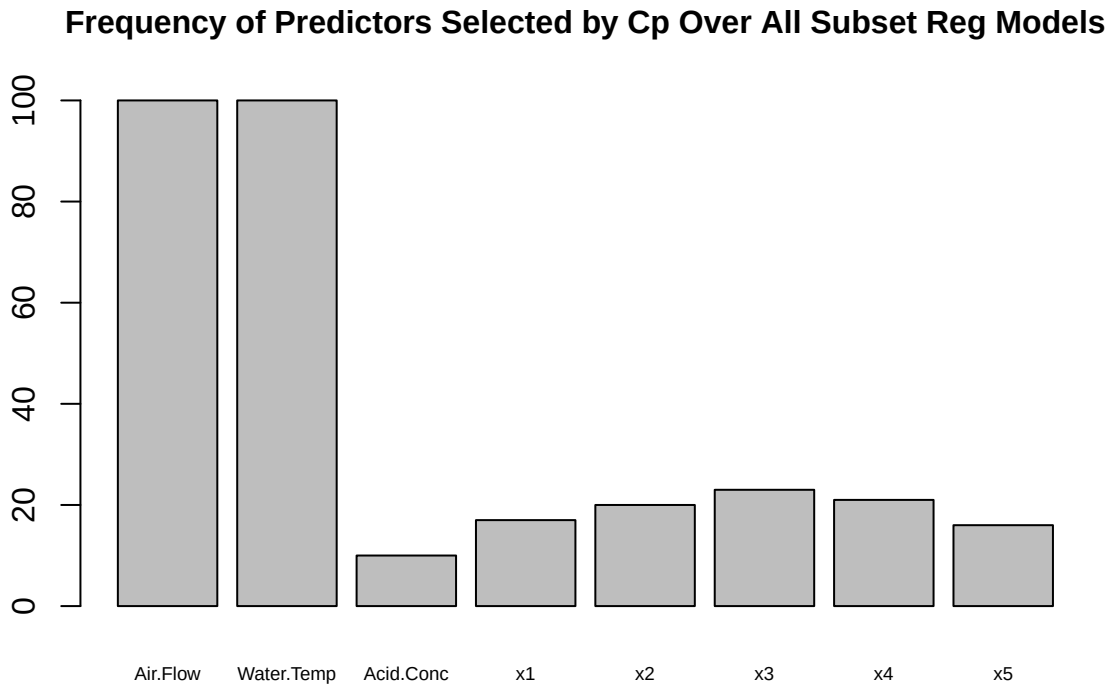
However, we do see from the plot of best AIC values vs. # of predictors that when the number of predictors is two, the AIC is quite close to the minimum AIC. In this case, we would have selected the same variables Air.Flow + Water.Temp.

Problem 4C

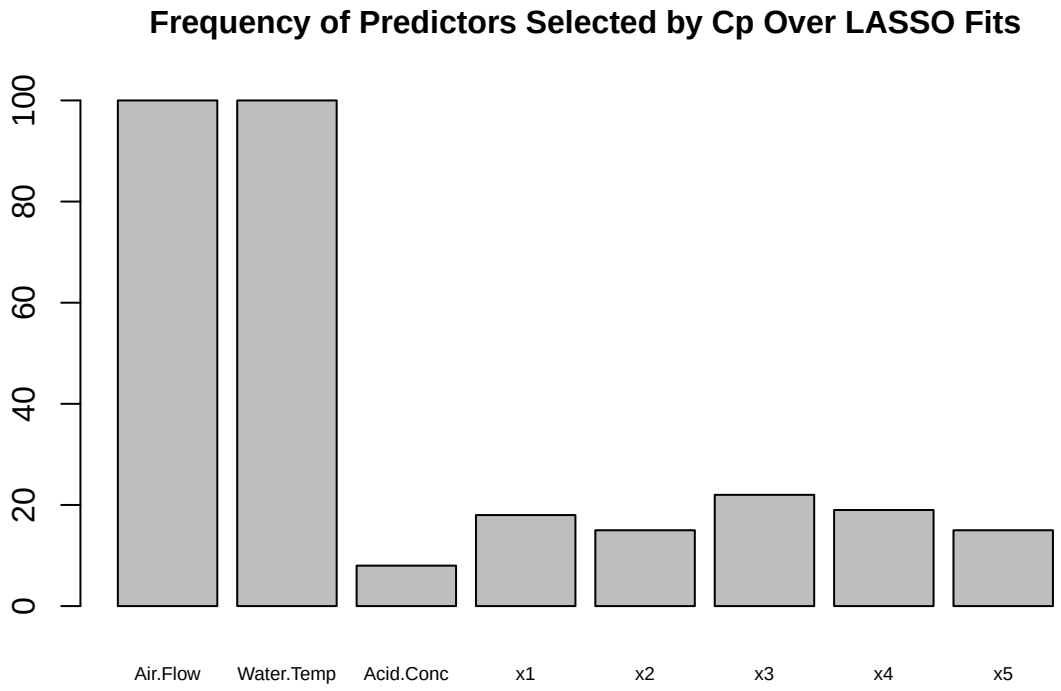
```
n = nrow(dat)
pred_count_leaps = rep(0, ncol(X))  # all subset lm
```

```
pred_count_lasso = rep(0, ncol(X))  # lasso
pred_count_bp = rep(0, ncol(X))  # BackwardPath
names(pred_count_leaps) = colnames(X)
names(pred_count_lasso) = colnames(X)
names(pred_count_bp) = colnames(X)

for(i in 1:100) {
  # generate junk variables and add to X
  x = matrix(runif(n*5, -1, 1), nrow=n)
  colnames(x) = c("x1", "x2", "x3", "x4", "x5")
  dat_junk = cbind(dat, x)
  X = dat_junk[, -which(names(dat_junk) %in% c("Stack.Loss"))]
  y = dat_junk$Stack.Loss

  # perform best subset regression model via Cp and record predictors selected
  L = leaps(X, y)
  ind = which.min(L$Cp)
  predictors = names(X)[L$which[ind,]]
  for(predictor in predictors) {
    pred_count_leaps[predictor] = pred_count_leaps[predictor] + 1
  }

  # perform best subset via LASSO
  fit = lm(y ~ ., data = X)
  lasso.fit = lars(x = model.matrix(fit)[,-1], y = y, normalize=FALSE, intercept=TRUE)
  actions = names(unlist(lasso.fit$actions))
  predictors = actions[1:(which.min(lasso.fit$Cp) - 1)]
  for(predictor in predictors) {
    pred_count_lasso[predictor] = pred_count_lasso[predictor] + 1
  }

  # perform backward path and record predictors selected
  coeffPath = BackwardPath(X, y)
  coefs = coeffPath$all_best_coefs[which.min(coeffPath$all_best_AICs)]
  predictors = names(unlist(coefs))[-1]
  for(predictor in predictors) {
    pred_count_bp[predictor] = pred_count_bp[predictor] + 1
  }
}
```

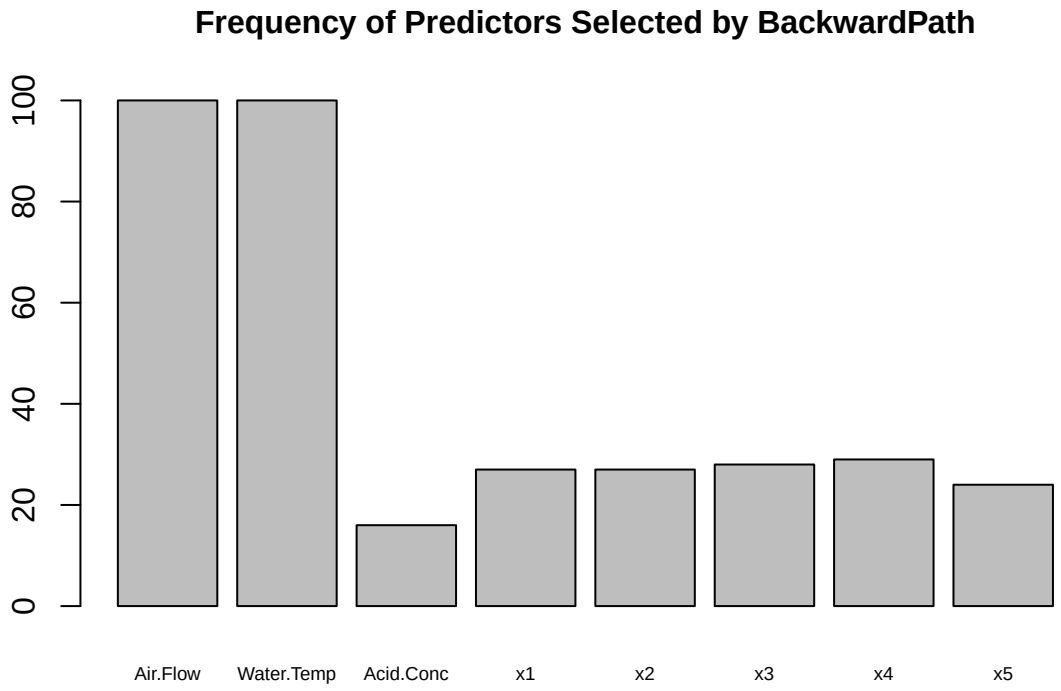
```
barplot(pred_count_leaps,
        main="Frequency of Predictors Selected by Cp Over All Subset Reg Models",
        cex.names=0.6, cex.main=1)
```




```
barplot(pred_count_lasso,
        main="Frequency of Predictors Selected by Cp Over LASSO Fits",
        cex.names=0.6, cex.main=1)
```




```
barplot(pred_count_bp,
        main="Frequency of Predictors Selected by BackwardPath",
        cex.names=0.6, cex.main=1)
```



All three methods include the junk variables in their "best" model at some point. The BackwardPath algorithm includes them the most at around 20-30% of the simulations and LASSO algorithm includes them the least at around 10-20% of the simulations. This makes sense, as LASSO penalization encourages coefficient estimates to go to zero, so it favors models with less variables.

All three methods include Air.Flow and Water.Temp every time, so although junk variables are included, the significant variables are never excluded.

We also see that none of the variables were left out completely; so all 8 variables were included in the "best" model at some point.