Tim Miller
15.071
HW #4

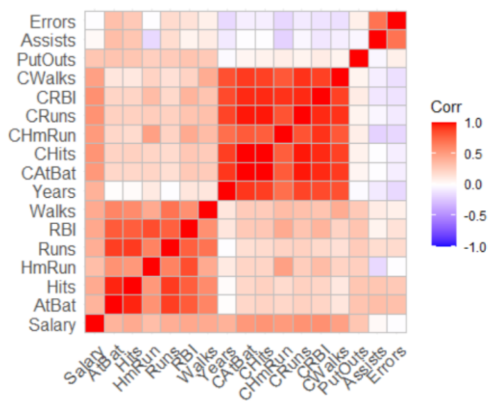
## Problem 1a

```r
#correlation matrix
matrix = cor(Hitters_raw[,2:18])
matrix
```

```
> matrix
             Salary     AtBat       Hits        HmRun        Runs        RBI     Walks       Years
Salary   1.000000000 0.3947709 0.43867474  0.343028078  0.41985856 0.44945709 0.4438673  0.40065699
AtBat    0.394770945 1.0000000 0.96396913  0.555102154  0.89982910 0.79601539 0.6244481  0.01272550
Hits     0.438674738 0.9639691 1.00000000  0.530627358  0.91063014 0.78847819 0.5873105  0.01859809
HmRun    0.343028078 0.5551022 0.53062736  1.000000000  0.63107588 0.84910743 0.4404537  0.11348842
Runs     0.419858559 0.8998291 0.91063014  0.631075883  1.00000000 0.77869235 0.6970151 -0.01197495
RBI      0.449457088 0.7960154 0.78847819  0.849107434  0.77869235 1.00000000 0.5695048  0.12966795
Walks    0.443867260 0.6244481 0.58731051  0.440453717  0.69701510 0.56950476 1.0000000  0.13479270
Years    0.400656994 0.0127255 0.01859809  0.113488420 -0.01197495 0.12966795 0.1347927  1.00000000
CAtBat   0.526135310 0.2071663 0.20667761  0.217463613  0.17181080 0.27812591 0.2694500  0.91568069
CHits    0.548909559 0.2253415 0.23560577  0.217495691  0.19132697 0.29213714 0.2707951  0.89784449
CHmRun   0.524930560 0.2124215 0.18936425  0.492525845  0.22970104 0.44218969 0.3495822  0.72237071
CRuns    0.562677711 0.2372778 0.23889610  0.258346846  0.23783121 0.30722616 0.3329766  0.87664855
CRBI     0.566965686 0.2213932 0.21938423  0.349858379  0.20233548 0.38777657 0.3126968  0.86380936
CWalks   0.489822036 0.1329257 0.12297073  0.227183183  0.16370021 0.23361884 0.4291399  0.83752373
PutOuts  0.300480356 0.3096075 0.29968754  0.250931497  0.27115986 0.31206456 0.2808555 -0.02001921
Assists  0.025436136 0.3421174 0.30397495 -0.161601753  0.17925786 0.06290174 0.1025226 -0.08511772
Errors  -0.005400702 0.3255770 0.27987618 -0.009743082  0.19260879 0.15015469 0.0819372 -0.15651196
              CAtBat       CHits      CHmRun       CRuns        CRBI      CWalks     PutOuts
Salary   0.526135310  0.54890956  0.52493056  0.56267771  0.56696569  0.48982204  0.30048036
AtBat    0.207166254  0.22534146  0.21242155  0.23727777  0.22139318  0.13292568  0.30960746
Hits     0.206677608  0.23560577  0.18936425  0.23889610  0.21938423  0.12297073  0.29968754
HmRun    0.217463613  0.21749569  0.49252584  0.25834685  0.34985838  0.22718318  0.25093150
Runs     0.171810798  0.19132697  0.22970104  0.23783121  0.20233548  0.16370021  0.27115986
RBI      0.278125914  0.29213714  0.44218969  0.30722616  0.38777657  0.23361884  0.31206456
Walks    0.269449974  0.27079505  0.34958216  0.33297657  0.31269680  0.42913990  0.28085548
Years    0.915680692  0.89784449  0.72237071  0.87664855  0.86380936  0.83752373 -0.02001921
CAtBat   1.000000000  0.99505681  0.80167609  0.98274694  0.95073014  0.90671165  0.05339251
CHits    0.995056810  1.00000000  0.78665204  0.98454184  0.94679739  0.89071842  0.06734799
CHmRun   0.801676089  0.78665204  1.00000000  0.82562483  0.92790264  0.81087827  0.09382223
CRuns    0.982746941  0.98454184  0.82562483  1.00000000  0.94567701  0.92776846  0.05908718
CRBI     0.950730141  0.94679739  0.92790264  0.94567701  1.00000000  0.88913701  0.09537515
CWalks   0.906711655  0.89071842  0.81087827  0.92776846  0.88913701  1.00000000  0.05816016
PutOuts  0.053392514  0.06734799  0.09382223  0.05908718  0.09537515  0.05816016  1.00000000
Assists -0.007897271 -0.01314420 -0.18888646 -0.03889509 -0.09655888 -0.06624345 -0.04339014
Errors  -0.070477521 -0.06803583 -0.16536941 -0.09408054 -0.11531613 -0.12993587  0.07530586
             Assists       Errors
Salary   0.025436136 -0.005400702
AtBat    0.342117377  0.325576978
Hits     0.303974950  0.279876183
HmRun   -0.161601753 -0.009743082
Runs     0.179257859  0.192608787
RBI      0.062901737  0.150154692
Walks    0.102522559  0.081937197
Years   -0.085117725 -0.156511957
CAtBat  -0.007897271 -0.070477521
CHits   -0.013144204 -0.068035829
CHmRun  -0.188886464 -0.165369407
CRuns   -0.038895093 -0.094080542
CRBI    -0.096558877 -0.115316131
CWalks  -0.066243445 -0.129935875
PutOuts -0.043390143  0.075305857
Assists  1.000000000  0.703504693
Errors   0.703504693  1.000000000
```

Plotting the matrix

```r
#plot correlation matrix
ggcorrplot(matrix)
```




The clear takeaway is that most of the statistics e.g., runs, homeruns, hits, at bats, etc. are a function of the amount of time (in years) the player has played in the MLB. Intuitively, this makes sense – we would expect that a player in the league for 15 year would have more hits than a player in the league for 5 years.

## Problem 1b

Statistics are useful for analyze player performance relative to other players. However, we do not want our analysis to be affected by the amount of time a player is in the league. By normalizing the data, we can put all the players on an equal footing to compare relative performance. So instead of just looking at raw numbers to predict performance, the normalized values allow us to understand relative performance. In other words, I would not want Player A on my team just because they have more total homeruns than Player B. Very likely Player A just has been in the league longer – this does not necessarily help predict what Player A will do the next year (e.g., Player A may be very old and not perform as well as a young up and coming Player B). Normalized values provide a more useful way to predict future performance.

## Problem 1c

Confirming mean salary of training set aligns:

```
> mean(y.train)
[1] 0.02788154
```

Aside from Names being removed from the dataset, it looks like the difference between x.train and train is that League, Division, and NewLeague columns have been changed from letters to 1s and 0s in the x.train matrix.

## Problem 1d

Create model

```
> summary(mod)

Call:
lm(formula = Salary ~ ., data = train)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.7873 -0.3736 -0.1048  0.2772  4.1992 
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.206659   0.098649   2.095 0.037708 *  
AtBat       -0.869866   0.249977  -3.480 0.000642 ***
Hits         0.999892   0.296046   3.377 0.000913 ***
HmRun       -0.115882   0.144532  -0.802 0.423836    
Runs        -0.145482   0.210373  -0.692 0.490197    
RBI         -0.018546   0.176755  -0.105 0.916563    
Walks        0.337571   0.104916   3.218 0.001556 ** 
Years       -0.111009   0.152005  -0.730 0.466244    
CAtBat       0.004799   0.799575   0.006 0.995219    
CHits        0.482802   1.207286   0.400 0.689742    
CHmRun       0.652609   0.405565   1.609 0.109499    
CRuns        0.238583   0.684242   0.349 0.727773    
CRBI        -0.123239   0.669537  -0.184 0.854188    
CWalks      -0.556092   0.221395  -2.512 0.012974 *  
PutOuts      0.146554   0.057281   2.559 0.011411 *  
Assists      0.103830   0.082155   1.264 0.208076    
Errors      -0.023454   0.075111  -0.312 0.755243    
LeagueN      0.072961   0.208689   0.350 0.727073    
DivisionW   -0.242129   0.108820  -2.225 0.027435 *  
NewLeagueN  -0.142724   0.209306  -0.682 0.496264    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6958 on 165 degrees of freedom
Multiple R-squared:  0.5967,	Adjusted R-squared:  0.5502 
F-statistic: 12.85 on 19 and 165 DF,  p-value: < 2.2e-16
```

We find the model has an $R^2$ of 0.5967.

Then we calculate $OSR^2$

```
#Calculate OSR-Squared
SSTTest = sum((test$Salary - mean(test$Salary))^2)
SSETest = sum((PredictTest - test$Salary)^2)
OSR2_LR <- 1 - SSETest/SSTTest
```

We find the model as $OSR^2$ of 0.1969206.

The $R^2$ value is higher than I expected. Based on my intuition, there are many young MLB players that have large salaries but have limited stats because of their limited time in the league. So perhaps that is why the $OSR^2$ is so low – absolute values don't necessarily help tell us how much the player is worth and thus how much they should be paid.

The significant variables are AtBat, Hits, Walks, Career Walks, Career Putouts, and Division. I would expect the coefficients of these variables to be positive. Intuitively, a player will be rewarded with a higher salary if they have more hits and walks – so makes sense those are positive. But surprising to see that Career Walks is negative. Interesting to see that AtBat has a negative coefficient. Perhaps this is because number of AtBats does not actually tell you anything about player performance e.g., they could get many AtBats but strikeout every time. Lastly, interesting that Division is significant and negative. My guess is that since there are differences in skill levels across teams in a certain division, the model is correcting for that. In other words, you might face bad teams in a certain division which is why you have good statistics, not because you might be a relatively better player.

## Problem 1e

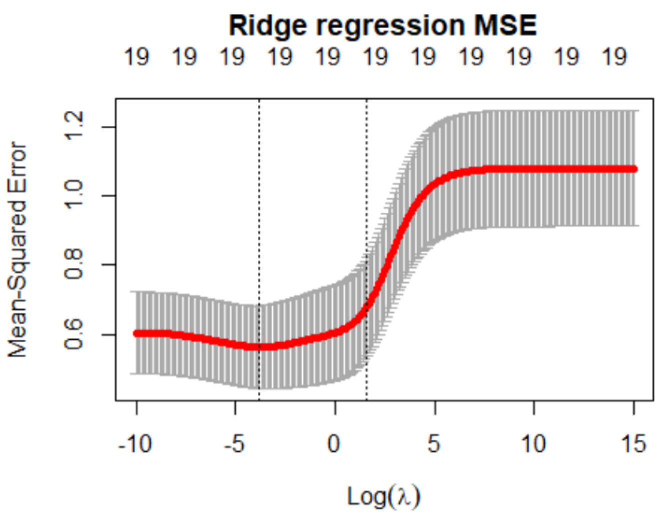
Both Ridge and LASSO help correct overfitting seen in simple linear regression. This is done by adding a penalty coefficient, lambda. The main difference between Ridge and LASSO is that LASSO helps perform some automatic feature selection as part of model development i.e., it will down select the most relevant features as it calculates final output.

Re: coefficients, we would expect smaller values for Ridge and LASSO vs the simple linear regression. Since these models look to prevent overfitting, they work to not put as much weight on specific coefficients.

## Problem 1f

Plot ridge regression:

```
# ridge regression cross-validation
all.lambdas <- c(exp(seq(15, -10, -.1)))
set.seed(300)
ridge.cv=cv.glmnet(x.train, y.train, alpha=0, lambda=all.lambdas)
plot(ridge.cv, main="Ridge regression MSE\n")
```




Find the lambda that delivers the lowest MSE:

```
#find min lambda
ridge.cv$lambda.min
```

The value of lambda that **minimizes mean squared error is 0.02237077**.

Re-train model using the best lambda:

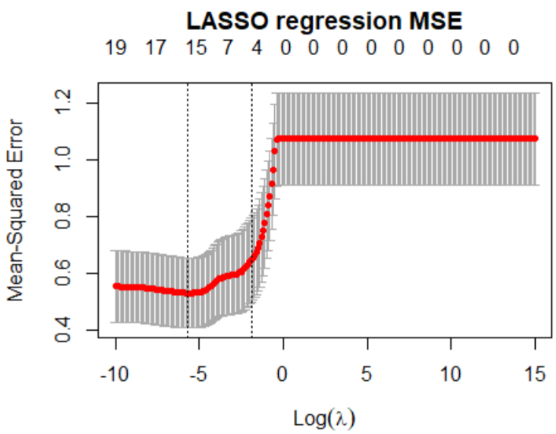
```
# ridge regression re-fit
best.lambda.ridge <- ridge.cv$lambda.min
ridge.model=glmnet(x.train,y.train,alpha=0,lambda=best.lambda.ridge)

beta.ridge = ridge.model$beta
sum(beta.ridge != 0)
```

We find that there are **19 coefficients are non-zero**.

## Problem 1g

Plot LASSO regression:

```
# LASSO cross-validation
all.lambdas <- c(exp(seq(15, -10, -.1)))
set.seed(301)
lasso.cv=cv.glmnet(x.train, y.train, alpha=1, lambda=all.lambdas)
plot(lasso.cv, main="LASSO regression MSE\n")
```




Find the lambda that delivers the lowest MSE:

```
#find min lambda
lasso.cv$lambda.min
```

The value of lambda that **minimizes mean squared error is 0.003345965**.

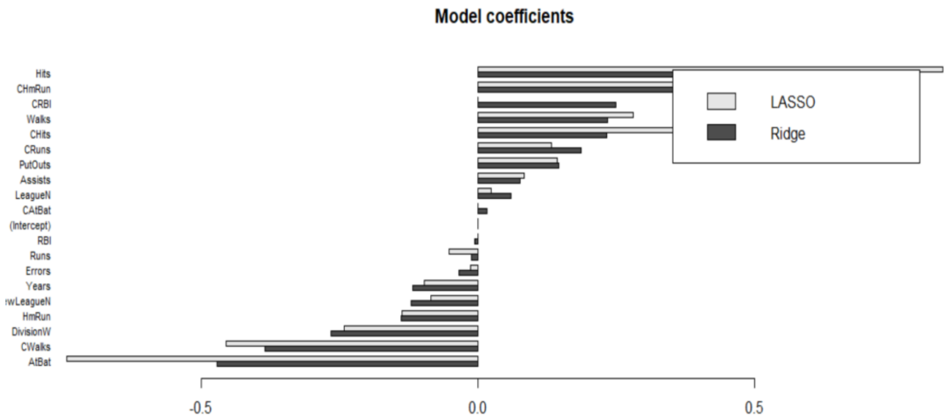
Re-train model using the best lambda:

```
# lasso regression re-fit
best.lambda.lasso <- lasso.cv$lambda.min
lasso.model=glmnet(x.train,y.train,alpha=1,lambda=best.lambda.lasso)

beta.lasso = lasso.model$beta
sum(beta.lasso != 0)
```

We find that there are **16 coefficients are non-zero**.

## Problem 1h

Plot of model coefficients:




There are a few notable differences between the models:

- Hits: much more important (e.g., higher positive value) in LASSO model for predicting salary
- Career RBIs: much more important (e.g., higher positive value) in Ridge model for predicting salary
- Career Hits: much more important (e.g., higher positive value) in LASSO model for predicting salary
- Career walks: both negative coefficients, but much more negative for LASSO model

There are similarities in coefficients across:

- Career Homeruns
- Walks, although LASSO slightly more positive
- PutOuts
- Assists
- Years

These coefficients align with my expectations. A player with a lot of Hits, Assists, and Putouts in the previous season and high career totals for Homeruns, RBI, Hits, Runs should expect to have a high salary. It shows both that they player performed well in the previous year and they have performed well over the course of their career. A little surprised that last season's RBI, Runs, and Homeruns are negative. One would expect that a player that performed well last year should perform well the next year. But I guess salaries are determined more by consistent performance over a number of years vs. relying too much on one year of data which might be an anomaly. Still, I would expect high numbers of RBI, Runs, and Homeruns to at least be somewhat positive.

Furthermore, I am not surprised to see Years and AtBats as negative. Intuitively you would pay older players less money because they are older and likely less productive due to their age.

One nuanced point. Years is negative, but career values for Homeruns, RBI, Hits, Runs are positive. This might seem counterintuitive. However, what we can say is that players that have high career totals i.e., have been consistently good tend to get rewarded with higher contracts. This consistent performance will outweigh the number of years they have played in the MLB. For middle of the pack players that have played a number of years but don't have high career totals, the Years value will have a bigger effect, as expected. You would not pay an aging average player a high salary.

# Problem 1i

**Ridge Model**

$R^2 = 0.5849596$
$OSR^2 = 0.2609523$

```
##RIDGE

# Make predictions on test and train sets
PredictTrain = predict(ridge.model, newx = x.train, s = best.lambda.ridge)
PredictTest = predict(ridge.model, newx = x.test, s = best.lambda.ridge)

# Calculate R-Squared
SSTTrain = sum((y.train - mean(y.train))^2)
SSETrain = sum((PredictTrain - y.train)^2)
R2_Ridge <- 1 - SSETrain/SSTTrain
R2_Ridge

#Calculate OSR-Squared
SSTTest = sum((y.test - mean(y.test))^2)
SSETest = sum((PredictTest - y.test)^2)
OSR2_Ridge <- 1 - SSETest/SSTTest
OSR2_Ridge
```

**LASSO Model**

$R^2 = 0.594256$
$OSR^2 = 0.2208032$

```
##LASSO

# Make predictions on test and train sets
PredictTrain = predict(lasso.model, newx = x.train, s = best.lambda.lasso)
PredictTest = predict(lasso.model, newx = x.test, s = best.lambda.lasso)

# Calculate R-Squared
SSTTrain = sum((y.train - mean(y.train))^2)
SSETrain = sum((PredictTrain - y.train)^2)
R2_LASSO <- 1 - SSETrain/SSTTrain
R2_LASSO

#Calculate OSR-Squared
SSTTest = sum((y.test - mean(y.test))^2)
SSETest = sum((PredictTest - y.test)^2)
OSR2_LASSO <- 1 - SSETest/SSTTest
OSR2_LASSO
```

The linear regression model with $R^2 = 0.5967$ performs better than both regularized models. However, both regularized models perform better than the linear regression on $OSR^2$.

The reason for this is that the regularized models correct for potential issues with overfitting. So likely the linear regression is overfit to the train data, causing it to perform worse on the test data. And that is why the regularized models perform better with $OSR^2$ – they are not as overfit to the train data.

## Problem 1j

```r
# Elastic net cross-validation
set.seed(302)
elnet.cv=train(Salary~.,train, method = "glmnet",
               trControl = trainControl(method = "cv", number = 10),
               tuneGrid=expand.grid(alpha=seq(0,1,.1), lambda=all.lambdas) )

#select best alpha and lambda
best.elnet.params=elnet.cv$bestTune
best_alpha = best.elnet.params$alpha
best_lambda = best.elnet.params$lambda

best_alpha
best_lambda
```

We find that the best alpha value is 0.6.
We find that the best lambda value is 0.04978707.

Given the alpha value of 0.6, I would expect the model to be more similar to the LASSO model.

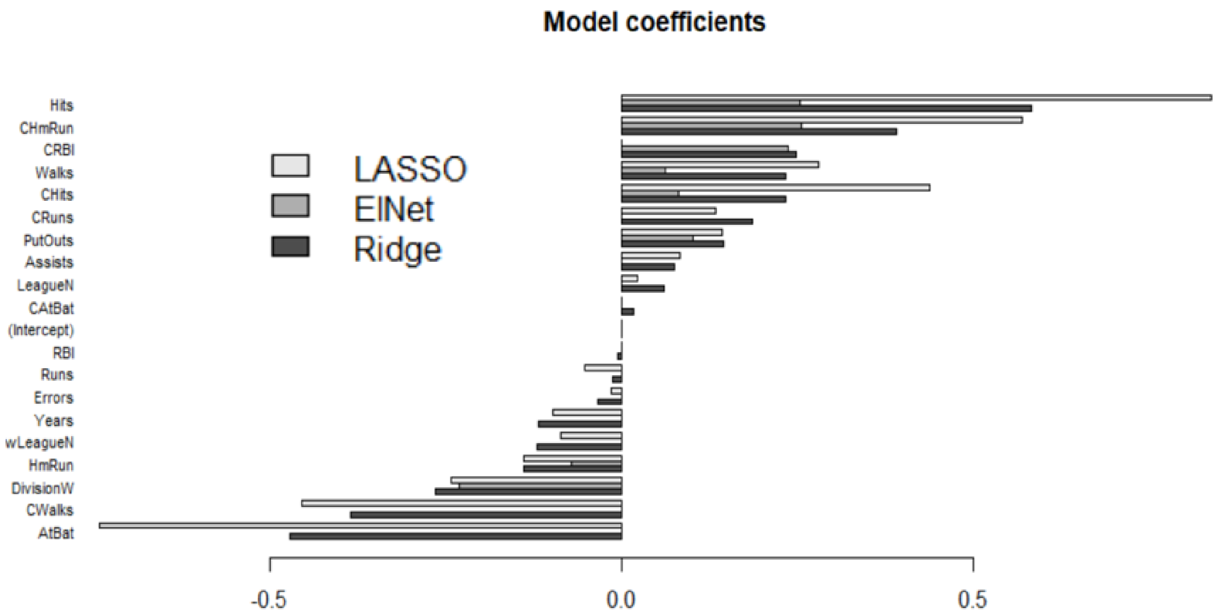
## Problem 1k

$R^2$ = 0.5215404
$OSR^2$ = 0.3042493

```r
##ELASTIC NET

# Make predictions on test and train sets
PredictTrain = predict(elnet.model, x.train)
PredictTest = predict(elnet.model, x.test)

# Calculate R-Squared
SSTTrain = sum((y.train - mean(y.train))^2)
SSETrain = sum((PredictTrain - y.train)^2)
R2_ELNET <- 1 - SSETrain/SSTTrain
R2_ELNET

#Calculate OSR-Squared
SSTTest = sum((y.test - mean(y.test))^2)
SSETest = sum((PredictTest - y.test)^2)
OSR2_ELNET <- 1 - SSETest/SSTTest
OSR2_ELNET
```

The model has a lower $R^2$ value than regression, Ridge, and LASSO. But compared to the other models it has the highest $OSR^2$.

## Problem 1l




**Some similarities**

- Hits, although absolute value of coefficient for Elnet is not as large as LASSO and Ridge
- Career Homeruns, although absolute value of coefficient for Elnet is not as large as LASSO and Ridge
- Career RBI – Elnet and Ridge are similar, LASSO has small (or no) coefficient value
- Division
- Putouts, although absolute value of coefficient for Elnet is not as large as LASSO and Ridge
- Homeruns, although absolute value of coefficient for Elnet is not as large as LASSO and Ridge

**Some differences**

- AtBat – no coefficient for Elnet
- CWalks – no coefficient for Elnet
- wLeagueN - no coefficient for Elnet
- Years – no coefficient for Elnet

What is interesting here is that RIDGE and LASSO are relatively similar, but Elnet is much different than both. This is surprising because I would have expected that this model would be the "average" of the other two models, but looks like that is not the case.

## Problem 1m

```
set.seed(308)
fs <- train(Salary~., train, method = "leapForward",
            trControl = trainControl(method = "cv", number = 10),
            tuneGrid = expand.grid(.nvmax=seq(1,15)))

fs$bestTune$nvmax

nvars.fs = fs$bestTune$nvmax
coef(fs$finalModel, nvars.fs)
```

Based on fs$bestTune$nvmax we found that the size of the best subset is 7.

Using coef(fs$finalModel, nvars.fs) we found that the 7 variables are:

- AtBat
- Hits
- Walks
- CRBI
- CWalks
- PutOuts
- DivisionW

Below are the coefficient values:

```
> coef(fs$finalModel, nvars.fs)
(Intercept)       AtBat        Hits       Walks        CRBI      CWalks     PutOuts   DivisionW
  0.1907450  -0.7599681   0.8427994   0.2384074   0.8073749  -0.3143548   0.1259589  -0.2685587
```

Yes this very much aligns with what we saw in 1h / 1l. In this updated model we see that AtBat, CWalks, and DivisionW are all negative coefficients – we see the same thing for LASSO and Ridge. Furthermore we see that Hits, Walks, CRBI, and PutOuts are all positive – we see the same thing for LASSO and Ridge.

## Problem 1n

For the updated model, we find:
$R^2$ = 0.5605935
$OSR^2$ = 0.3057758

$R^2$ value is slightly less than the original model, but we have a much higher $OSR^2$.

```
# Make predictions on test and train sets
PredictTrain = predict(fs, newdata = train)
PredictTest = predict(fs, newdata = test)

# Calculate R-Squared
SSTTrain = sum((train$Salary - mean(train$Salary))^2)
SSETrain = sum((PredictTrain - train$Salary)^2)
R2_LR2 <- 1 - SSETrain/SSTTrain
R2_LR2

#Calculate OSR-Squared
SSTTest = sum((test$Salary - mean(test$Salary))^2)
SSETest = sum((PredictTest - test$Salary)^2)
OSR2_LR2 <- 1 - SSETest/SSTTest
OSR2_LR2
```

## Problem 1o

```r
# XGBoost
set.seed(304)
xgb.cv <-  train(Salary~.,train,
                method = "xgbTree", objective = "reg:squarederror",
                trControl = trainControl(method="cv", number=10))

best.xgb.params=xgb.cv$bestTune
best.xgb.params
```

Using best.xgb.params we find the following best parameters for XGBoost:

```
> best.xgb.params
   nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
28      50         2 0.3     0              0.8                1       0.5
```

## Problem 1p

Create model (note: made alpha and lambda both 1 based on the R code from lecture 9…there was no alpha or lambda value in the best.xgb.params output.

```r
xgboost.model = xgboost(data = x.train,
                        label = y.train,
                        params = list(alpha=1,
                                      colsample_bytree = 0.8,
                                      eta = 0.3,
                                      gamma = 0,
                                      lambda = 1,
                                      max_depth = 2,
                                      min_child_weight=1,
                                      subsample = .5),
                        nrounds = 50,
                        verbose = F)
```

```
# Make predictions on test and train sets
PredictTrain = predict(xgboost.model, x.train)
PredictTest = predict(xgboost.model, x.test)

# Calculate R-Squared
SSTTrain = sum((y.train - mean(y.train))^2)
SSETrain = sum((PredictTrain - y.train)^2)
R2_XGB <- 1 - SSETrain/SSTTrain
R2_XGB

#Calculate OSR-Squared
SSTTest = sum((y.test - mean(y.test))^2)
SSETest = sum((PredictTest - y.test)^2)
OSR2_XGB <- 1 - SSETest/SSTTest
OSR2_XGB
```

For the XGBoost we find the $R^2$ is a staggering 0.9274611 and the $OSR^2$ is 0.5395549. Unsurprisingly, this performs better than all the other models we tested.

The advantages for this model is that we have a higher $R^2$ and higher $OSR^2$. The biggest disadvantage is the amount of time it takes to run. This dataset is relatively small and it took much longer to get to the model output. Furthermore, this is a very "black box" model and would be hard to explain to any non-analytical audience.

## Problem 1q

Given the sheer number of iterations run using XG Boost, it can smooth out some of the randomness that the linear regression models are not able to do. The linear regression models simply just are not able to test as many iterations on the data.