Project

Kaleb Williams

2024-03-21

OLS

For our predictions we shall predict the average home price in vancouver based of various predictors using lasso, ridge and regular linear regression. This can also be partioned into average price oh homes with "blank" bedrooms for example for a more in depth analysis.

```r
data <- read.csv('Housing.csv')
```

Data is a little messy will have to convert all of the yes, no answers for predictors such that they are binary 0,1. Therefore for a yes reply 1 will be attributed and for a no answer 0 will be attributed.

Ultimately we wish to predict the housing price using predictors from the data set. The area of the home, number of bedrooms, bathrooms, if there is air conditioning, if there is hot water heating, parking and if there is a basement will be chosen as predictors for the regressions. As these will likely impact the price of the home the most.

Price and area are continuous variables while number of bedrooms/bathrooms and parking are categorical so we should find out the min/max number of each.

```r
#Finding the min/max number of bedrooms, bathrooms and parking spots

max(data$bedrooms)
```

```
## [1] 6
```

```r
max(data$bathrooms)
```

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

```r
max(data$parking)
```

```
## [1] 3
```

```r
min(data$bedrooms)
```

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

```r
min(data$bathrooms)
```

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

```r
min(data$parking)
```

```
## [1] 0
```

Therefore the number of bedrooms is categorical with values 1:6, number of bathrooms is categorical with values 1:4 and number of parking spots is categorical with values 0:3.

```r
#Extracting necessary columns from data

extracted_columns <- data.frame("Price"=c(data$price),
                                "Area" = c(data$area),
                                "Bedroom" = c(data$bedrooms),
                                "Bathroom" = c(data$bathrooms),
                                "HotWater"=c(data$hotwaterheating),
                                "Air"=c(data$airconditioning),
                                "Parking"=c(data$parking))
```

```r
#Converting yes/no answers into binary responses

extracted_columns$HotWater = ifelse(extracted_columns$HotWater == "yes",1,0)

extracted_columns$Air = ifelse(extracted_columns$Air == "yes",1,0)
```

Since the data is scaled quite differently for price and area we should standardize the variables. Before standardizing we will save the dataframe.

```r
#Saving data frame

original_data <- extracted_columns
```

```r
#Standardizing the Price and Area variables

extracted_columns$Price <- scale(extracted_columns$Price)

extracted_columns$Area <- scale(extracted_columns$Area)

head(extracted_columns)
```

```
##      Price     Area Bedroom Bathroom HotWater Air Parking
## 1 4.562174 1.045766       4        2        0   1       2
## 2 4.000809 1.755397       4        4        0   1       3
## 3 4.000809 2.216196       3        2        0   0       2
## 4 3.982096 1.082630       4        2        0   1       3
## 5 3.551716 1.045766       4        1        0   1       2
## 6 3.252321 1.082630       3        3        0   1       2
```

We will use the caret package to partition the data randomly into 80% training set and 20% testing set.

```r
library(caret)

# Set the seed for reproducibility
set.seed(123)

# Split the dataframe into train and test sets
train_indices <- createDataPartition(extracted_columns$Price, p = 0.8, list = FALSE)
train_data <- extracted_columns[train_indices, ]
test_data <- extracted_columns[-train_indices, ]

# Show the sizes of train and test sets
cat("Training set size:", nrow(train_data), "\n")
```

```
## Training set size: 438
```

```r
cat("Testing set size:", nrow(test_data), "\n")
```

```
## Testing set size: 107
```

Now that we have the training and test data we can apply a simple linear regression to the data using all the predictors chosen.

```r
#Training a basic linear regression model

linreg_model <- lm(Price ~.,data=train_data)

summary(linreg_model)
```

```
##
## Call:
## lm(formula = Price ~ ., data = train_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.66185 -0.39333 -0.02023  0.34956  2.76395
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.67645    0.13782 -12.164  < 2e-16 ***
## Area         0.34728    0.03341  10.395  < 2e-16 ***
## Bedroom      0.17502    0.04503   3.887 0.000118 ***
## Bathroom     0.64341    0.06713   9.585  < 2e-16 ***
## HotWater     0.38228    0.14028   2.725 0.006690 **
## Air          0.58723    0.06978   8.415 5.85e-16 ***
## Parking      0.16589    0.03883   4.272 2.38e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6416 on 431 degrees of freedom
## Multiple R-squared:  0.577,  Adjusted R-squared:  0.5711
## F-statistic: 97.97 on 6 and 431 DF,  p-value: < 2.2e-16
```

From the summary we can see that each of the predictors is greatly significant in to the price response. Now to make predictions with this model.

```
#Predicting using the test data

OLR_prediction <- predict(linreg_model,newdata=test_data)

head(OLR_prediction)
```

```
##          1          2          7         13         17         21
## 1.59264148 3.29178334 2.42167694 1.28752873 1.29553000 0.07318709
```

```
#Calculating accuracy metrics

data.frame(
  RMSE = RMSE(OLR_prediction, test_data$Price),
  Rsquare = R2(OLR_prediction,test_data$Price),
  MAE = MAE(OLR_prediction,test_data$Price)
)
```

```
##        RMSE   Rsquare       MAE
## 1 0.6653989 0.6219545 0.4840322
```

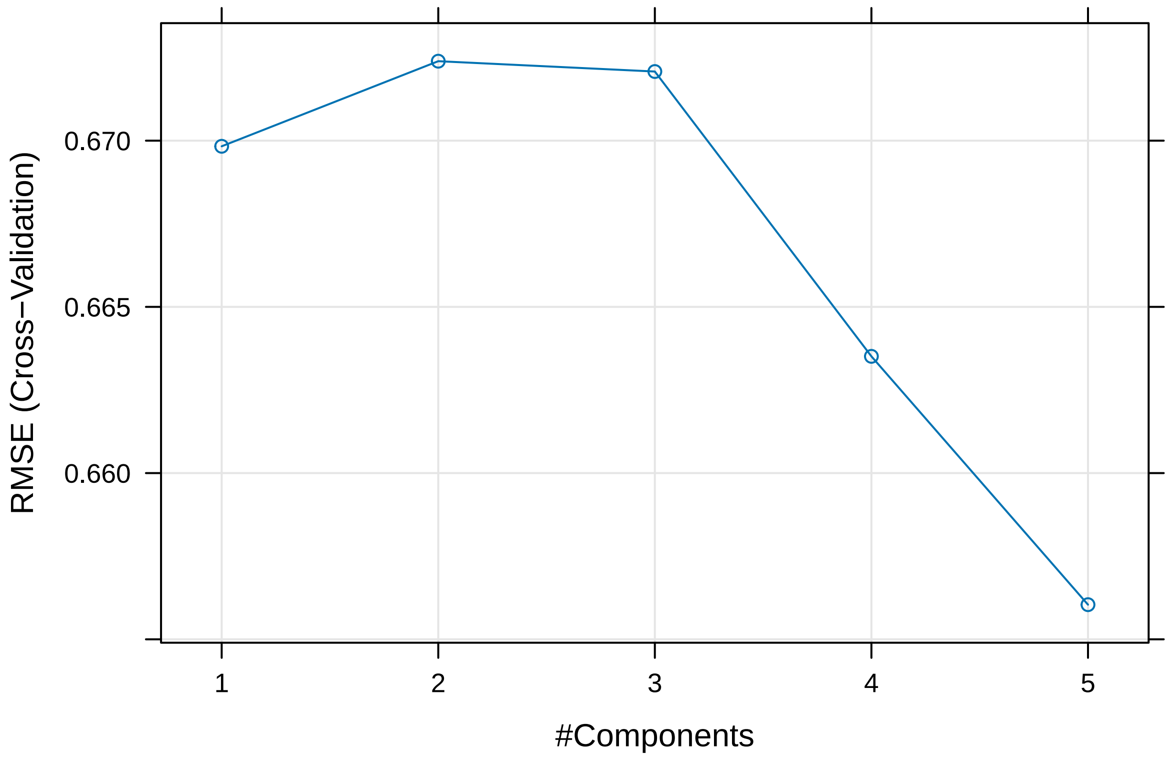
Overall very low error for the predictive model on the test data.

Now we can use 10-fold cross validation in order to determine best possible splitting of the data using the caret package once again.

```
#10 fold cross validation
library(caret)

set.seed(1234)

model1 <- train(
  Price~.,data=extracted_columns,method='pcr',
  scale =TRUE,
  trControl = trainControl(method='cv',number =10),
  tuneLength=10
)

model1$resample
```

```
##         RMSE  Rsquared       MAE Resample
## 1  0.6962130 0.5764652 0.4884428   Fold09
## 2  0.6186707 0.6916845 0.4774611   Fold04
## 3  0.7319539 0.5847734 0.5191504   Fold06
## 4  0.5929023 0.5851957 0.4754581   Fold01
## 5  0.6474938 0.5136166 0.5037032   Fold02
## 6  0.7103326 0.5593800 0.5054457   Fold10
## 7  0.6223134 0.5027904 0.4894592   Fold05
## 8  0.6691130 0.6192489 0.5488835   Fold07
## 9  0.6781445 0.4403796 0.5532232   Fold08
## 10 0.5932845 0.7108709 0.4608347   Fold03
```

```r
plot(model1)
```



This plot tells us that choosing 5 principle components (regressors) will yield the lowest RMSE value for the regression.

```r
#Making predictions using this best model

cv_predictions <- model1 |>
  predict(test_data)

OLR_accuracy <- data.frame(
  RMSE =RMSE(cv_predictions,test_data$Price),
  Rsquare = R2(cv_predictions,test_data$Price),
  MAE = MAE(cv_predictions,test_data$Price)
)
```

RMSE from the original model is 0.6653989 and the RMSE from the 10 fold cross validated model is 0.6630291. Slightly better but not by much at all, showing the original model was quite accurate in predicting the Vancouver housing prices.

LASSO

Least absolute selection and shrinkage operator (LASSO) is an updated regression applying an additive term to the regular least squares estimate for the parameters of the regression and as before of ordinary least squares minimizing its cost function. Mathematically defined as

$$\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{n}\beta_o + \beta_1 x_1 + \beta_2 x_2 + \dots \beta_p x_p\right)^2 + \lambda\sum_{j=1}^{n}\beta_j^2. \tag{1}$$

The penalty is calculated as the Manhatten distance which is defined as

$$d(\mathbf{X},\mathbf{Y}) = |x_1 - x_2| + |y_1 - y_2| \tag{2}$$

It involves finding the optimal value of lambda to scale the parameters to optimal solutions. That is it applies a penalty to the parameters that are too small and applies nothing to the parameters that are within optimal range. This encourages sparsity in the model. Sparsity, many of the coefficents are equal to zero, is desirable for many reasons.

Feature Selection: When many coefficients are zero, it indicates that the corresponding predictors (features) have little to no influence on the response variable. This effectively performs feature selection, simplifying the model by removing irrelevant or redundant predictors.

Interpretability: A sparse model is easier to interpret because it focuses only on a subset of predictors that have significant impact on the response variable. With fewer predictors to consider, it's easier to understand the relationships between predictors and the response.

Reduced Overfitting: Sparsity helps prevent overfitting, where the model captures noise in the training data rather than the underlying patterns. By reducing the number of predictors, sparse models are less likely to overfit and generalize better to unseen data.

Computational Efficiency: Sparse models with fewer nonzero coefficients require less computational resources for both training and inference. This makes sparse models particularly useful for large-scale datasets and real-time applications where efficiency is important.

Improved Stability: Sparse models tend to be more stable because they rely on fewer predictors, which reduces the impact of small changes in the data or slight variations in the training process.

Ultimately we want to find the optimal value of λ that reduces the redundant predictors to zero or having very little impact on the model. This is typically done through cross validation.LASSO can be performed using the glmnet package. For LASSO we will use all variables contained within the data set.

We will now need to clean all the data applying the same binary rule as stated in the OLR section. The only thing that will differ is the furnishingstatus column we shall also apply a categorical conversion to this variable. 2 for furnished, 1 for semi-furnished and 0 for unfurnished.

```r
#Cleaning data for LASSO

LASSO_data <- data.frame("Price"=c(data$price),
"Area" =c(data$area),
"Bedroom" = c(data$bedrooms),
"Bathroom" = c(data$bathrooms),
"Stories" = c(data$stories),
"Main" = c(data$mainroad),
"Guest" = c(data$guestroom),
"Basement" = c(data$basement),
"HotWater"=c(data$hotwaterheating),                "Air"=c(data$airconditioning),
"Parking"=c(data$parking),
"Pre"=c(data$prefarea),
"Furnished"=c(data$furnishingstatus))
```

```r
#Converting all binary and categorical variables
LASSO_data$Main  = ifelse(LASSO_data$Main == "yes",1,0)

LASSO_data$Guest = ifelse(LASSO_data$Guest == "yes",1,0)

LASSO_data$Basement  = ifelse(LASSO_data$Basement == "yes",1,0)

LASSO_data$HotWater  = ifelse(LASSO_data$HotWater == "yes",1,0)

LASSO_data$Air  = ifelse(LASSO_data$Air == "yes",1,0)

LASSO_data$Pre = ifelse(LASSO_data$Pre == "yes",1,0)

LASSO_data$Furnished = ifelse(LASSO_data$Furnished == "furnished", 2,
                              ifelse(LASSO_data$Furnished == "semi-furnished", 1, 0))
```

```r
#Showing converted data frame
head(LASSO_data)
```

```
##      Price Area Bedroom Bathroom Stories Main Guest Basement HotWater Air
## 1 13300000 7420       4        2       3    1     0        0        0   1
## 2 12250000 8960       4        4       4    1     0        0        0   1
## 3 12250000 9960       3        2       2    1     0        1        0   0
## 4 12215000 7500       4        2       2    1     0        1        0   1
## 5 11410000 7420       4        1       2    1     1        1        0   1
## 6 10850000 7500       3        3       1    1     0        1        0   1
##   Parking Pre Furnished
## 1       2   1         2
## 2       3   0         2
## 3       2   1         1
## 4       3   1         2
## 5       2   0         2
## 6       2   1         1
```

Now to apply LASSO we must create a linear regression will all variables as predictors. Just like with OLR we shall standardize the price and area columns.

```r
#Standardizing the price and area columns

LASSO_data$Price <- scale(LASSO_data$Price)

LASSO_data$Area <- scale(LASSO_data$Area)

head(LASSO_data)
```

```
##      Price     Area Bedroom Bathroom Stories Main Guest Basement HotWater Air
## 1 4.562174 1.045766       4        2       3    1     0        0        0   1
## 2 4.000809 1.755397       4        4       4    1     0        0        0   1
## 3 4.000809 2.216196       3        2       2    1     0        1        0   0
## 4 3.982096 1.082630       4        2       2    1     0        1        0   1
## 5 3.551716 1.045766       4        1       2    1     1        1        0   1
## 6 3.252321 1.082630       3        3       1    1     0        1        0   1
##   Parking Pre Furnished
## 1       2   1         2
## 2       3   0         2
## 3       2   1         1
## 4       3   1         2
## 5       2   0         2
## 6       2   1         1
```

Now we must split the data once again into training and testing using the caret package.

```
library(caret)

# Set the seed for reproducibility
set.seed(123)

# Split the dataframe into train and test sets
train_indices <- createDataPartition(LASSO_data$Price, p = 0.8, list = FALSE)
LASSO_train_data <- LASSO_data[train_indices, ]
LASSO_test_data <- LASSO_data[-train_indices, ]

# Show the sizes of train and test sets
cat("Training set size:", nrow(LASSO_train_data), "\n")
```

```
## Training set size: 438
```

```
cat("Testing set size:", nrow(LASSO_test_data), "\n")
```

```
## Testing set size: 107
```

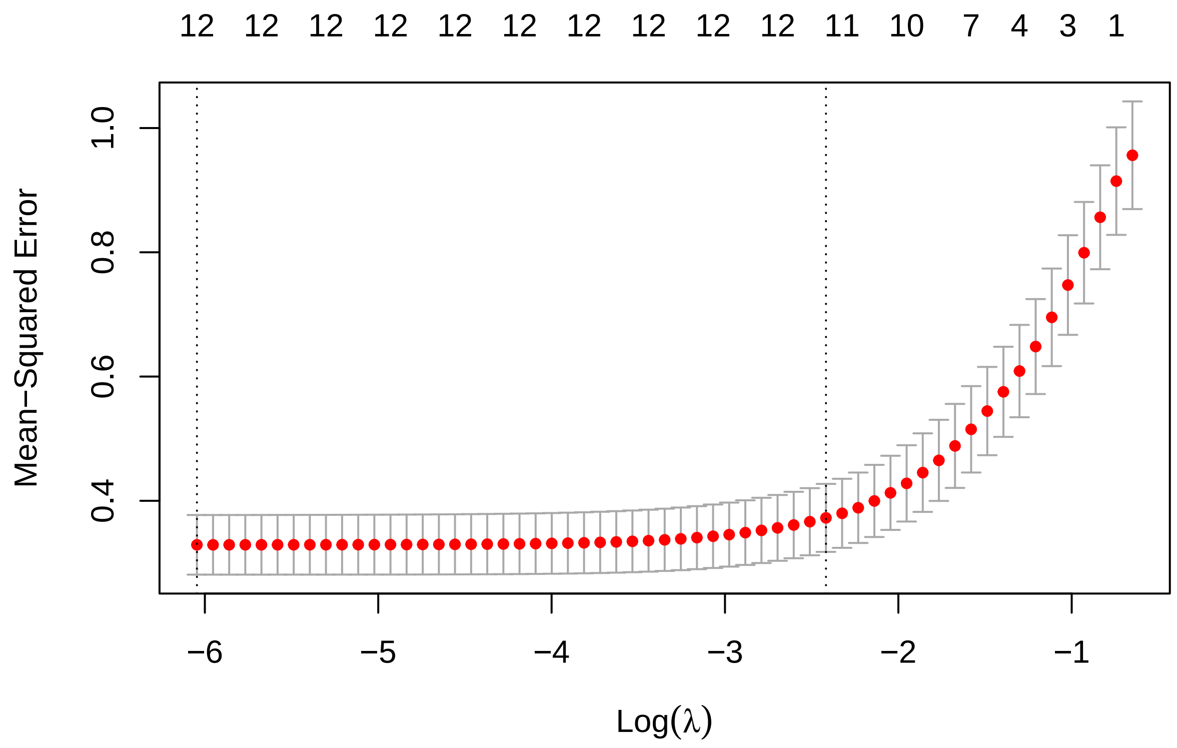
```
#Creating matrix for training data to create LASSO model

#Training data
x1 = as.matrix(LASSO_train_data[,2:13])
y1 = as.matrix(LASSO_train_data[,1])
```

```
#Applying LASSO regression to the data

library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
LASSO_train_model = cv.glmnet(x1,y1,alpha=1,family="gaussian")
plot(LASSO_train_model)
```



Therefore small values of λ will be preferable for the LASSO regression.

```
#Finding minimum \lambda

LASSO_train_model$lambda.min
```

```
## [1] 0.002367939
```

Now to make predictions with the LASSO model and the training data.

```
#Test data matrix

x2 = as.matrix(LASSO_test_data[,2:13])

LASSO_predictions <- predict(LASSO_train_model, newx = x2)
```

```
#Calculating accuracy metrics

LASSO_accuracy <- data.frame(
  RMSE = RMSE(LASSO_predictions, LASSO_test_data$Price),
  Rsquare = R2(LASSO_predictions,LASSO_test_data$Price),
  MAE = MAE(LASSO_predictions,LASSO_test_data$Price)
)

LASSO_accuracy
```

```
##        RMSE lambda.1se       MAE
## 1 0.6540647  0.6594027 0.4682349
```

Ridge

Ridge regression is essentially the same as LASSO except for that instead of using the Manhattan distance for the computation of the penalty, it using the standard Euclidean distance.

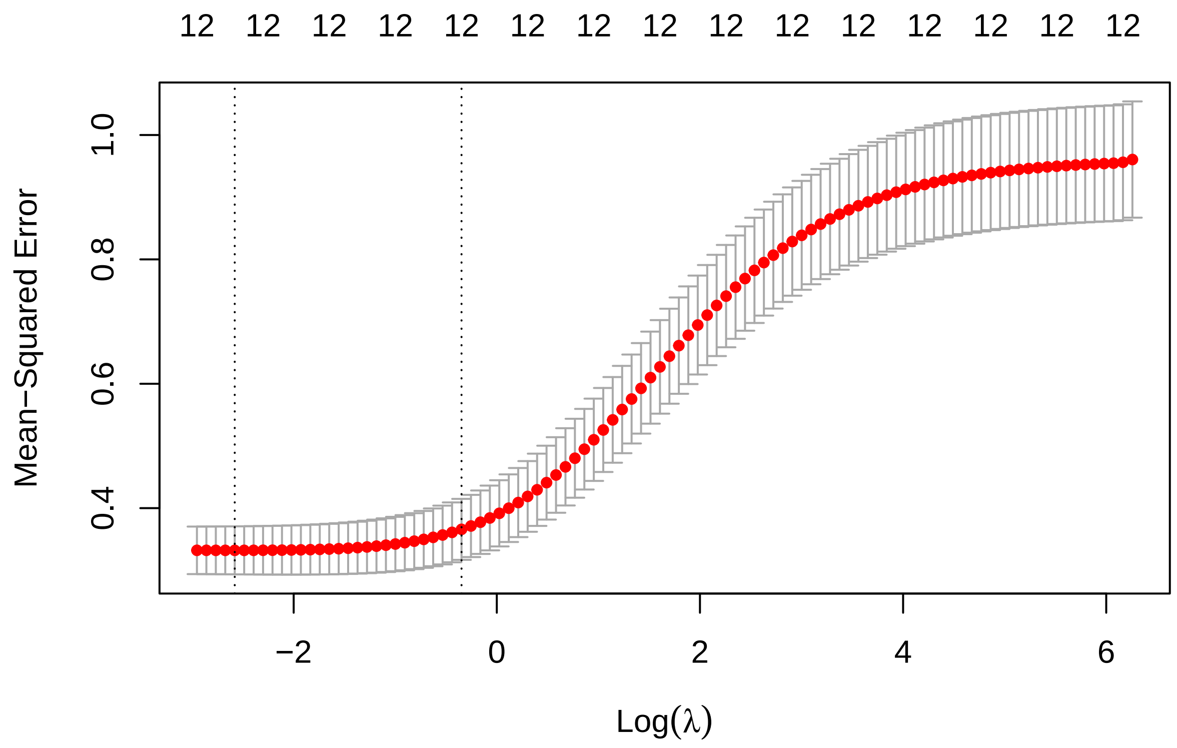
$$d(\mathbf{X}, \mathbf{y}) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3}$$

The exact same methodology and reasoning apply to Ridge as LASSO and again can be calculated using the glmnet package.

Since ridge requires all the data as regressors we can relabel the variables from the LASSO portion.We only need the training and testing portions of the data.

```
#Relabeling LASSO training and test data

#Training
x1_ridge = x1
y1_ridge = y1

#Testing
x2_ridge = x2
```

```
#Creating linear model using the test data

library(glmnet)

ridge_train_model = cv.glmnet(x1_ridge,y1_ridge,alpha=0,family="gaussian",nlamnda=1e2)
plot(ridge_train_model)
```



Somewhat similar results to LASSO.

```
#Optimal lambda value

ridge_train_model$lambda.min
```

```
## [1] 0.07575668
```

Much larger value of λ then LASSO regression.

Now to make our ridge predictions.

```
#Ridge predictions

ridge_predictions <- predict(ridge_train_model, newx = x2_ridge)
```

```
#Calculating accuracy metrics

Ridge_accuracy <- data.frame(
  RMSE = RMSE(ridge_predictions, LASSO_test_data$Price),
  Rsquare = R2(ridge_predictions,LASSO_test_data$Price),
  MAE = MAE(ridge_predictions,LASSO_test_data$Price)
)

Ridge_accuracy
```

```
##        RMSE lambda.1se       MAE
## 1 0.6564388  0.6716484 0.4707475
```

```
#Comparing accuracies

acc_df <- data.frame( OLR = OLR_accuracy, LASSO = LASSO_accuracy, Ridge = Ridge_accuracy)

acc_df
```

```
##    OLR.RMSE OLR.Rsquare  OLR.MAE LASSO.RMSE LASSO.lambda.1se LASSO.MAE
## 1 0.6630291   0.6233139 0.485016  0.6540647        0.6594027 0.4682349
##   Ridge.RMSE Ridge.lambda.1se Ridge.MAE
## 1  0.6564388        0.6716484 0.4707475
```

Although very close in value LASSO produces the greatest results for accuracy for all RMSE and MAE.