MSCI 446 - Assignment 2 - Question 1

Mark Harper, Hossam Gomaa, Kenta Morris

22/02/2021

Include Packages

```
library('tidyverse')
library('caret')
library('gridExtra')
library('plotly')
library('ISLR')
library('ggplot2')
library('gridExtra')
library('glmnet')

theme_set(theme_classic())
```

Question 1: Nonlinear Regression

#1.1 Process your data

We will use a dataset of used cars being sold on a website called Cardekho.

```
data <- read.csv('cars_cardekho.csv')
nrow(data)
```

```
## [1] 8128
```

```
summary(is.na(data))
```

```
##     name              year          selling_price     km_driven
##  Mode :logical    Mode :logical    Mode :logical    Mode :logical
##  FALSE:8128       FALSE:8128       FALSE:8128       FALSE:8128
##
##     fuel          seller_type      transmission       owner
##  Mode :logical    Mode :logical    Mode :logical    Mode :logical
##  FALSE:8128       FALSE:8128       FALSE:8128       FALSE:8128
##
##   mileage           engine          max_power         torque
##  Mode :logical    Mode :logical    Mode :logical    Mode :logical
##  FALSE:8128       FALSE:8128       FALSE:8128       FALSE:8128
##
```

```
##    seats
##  Mode :logical
##  FALSE:7907
##  TRUE :221
```

It appears as though there are 221 missing entries in the seat column. We will remove these 221 data points, than downsample the 8000+ data entries to a more workable 4500, than use str() to summarize the columns.

```r
data <- na.omit(data)
nrow(data)
```

```
## [1] 7907
```

```r
data <- data[sample(4500),]
str(data)
```

```
## 'data.frame':    4500 obs. of  13 variables:
##  $ name         : chr  "Hyundai EON Era Plus Sports Edition" "Maruti A-Star Lxi" "Ford Fiesta EXi 1.4
##  $ year         : int  2017 2010 2011 2017 2016 2019 2013 2016 2014 2018 ...
##  $ selling_price: int  350000 235000 275000 775000 350000 5150000 750000 875000 605000 395000 ...
##  $ km_driven    : int  20000 50000 120000 32000 25000 20000 79328 40000 80000 10800 ...
##  $ fuel         : chr  "Petrol" "Petrol" "Diesel" "Diesel" ...
##  $ seller_type  : chr  "Individual" "Individual" "Individual" "Dealer" ...
##  $ transmission : chr  "Manual" "Manual" "Manual" "Manual" ...
##  $ owner        : chr  "Second Owner" "First Owner" "Second Owner" "First Owner" ...
##  $ mileage      : chr  "21.1 kmpl" "19.0 kmpl" "17.8 kmpl" "24.3 kmpl" ...
##  $ engine       : chr  "814 CC" "998 CC" "1399 CC" "1248 CC" ...
##  $ max_power    : chr  "55.2 bhp" "66.1 bhp" "68 bhp" "88.5 bhp" ...
##  $ torque       : chr  "74.5Nm@ 4000rpm" "90Nm@ 3500rpm" "16.3@ 2,000(kgm@ rpm)" "200Nm@ 1750rpm" ..
##  $ seats        : int  5 5 5 5 5 5 7 5 7 5 ...
##  - attr(*, "na.action")= 'omit' Named int [1:221] 14 32 79 88 120 139 201 207 229 253 ...
##   ..- attr(*, "names")= chr [1:221] "14" "32" "79" "88" ...
```

We will attempt to predict the *selling_price* of a used car being sold based on the following input characteristics:

- *year* (numeric)
- *km_driven* (numeric)
- *max_power* (numeric)
- *torque* (numeric)
- *transmission* (categorical)

Note that *max_power* and *torque* inputs are currently **chr** when they are actually **num** in nature. We will need to remove any characters from the data entries and convert to numeric variables before proceeding.

```r
target.cols <- c('selling_price', 'year', 'km_driven', 'max_power', 'torque', 'transmission')
data <- data[, target.cols]

data$max_power <- as.numeric(gsub(' bhp', '', data$max_power))
data$torque <- as.numeric(gsub('\\Nm.*', '', data$torque))
```

```
## Warning: NAs introduced by coercion
```

```r
#check for missing values:
summary(is.na(data))
```

```
##  selling_price      year         km_driven       max_power
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:4500      FALSE:4500      FALSE:4500      FALSE:4500
##
##    torque        transmission
##  Mode :logical   Mode :logical
##  FALSE:4088      FALSE:4500
##  TRUE :412
```

Drop the missing data points (412 missing entries for torque that will confuse our ML algorithms)

```r
data <- na.omit(data)
```

1.2 Train / Test Split

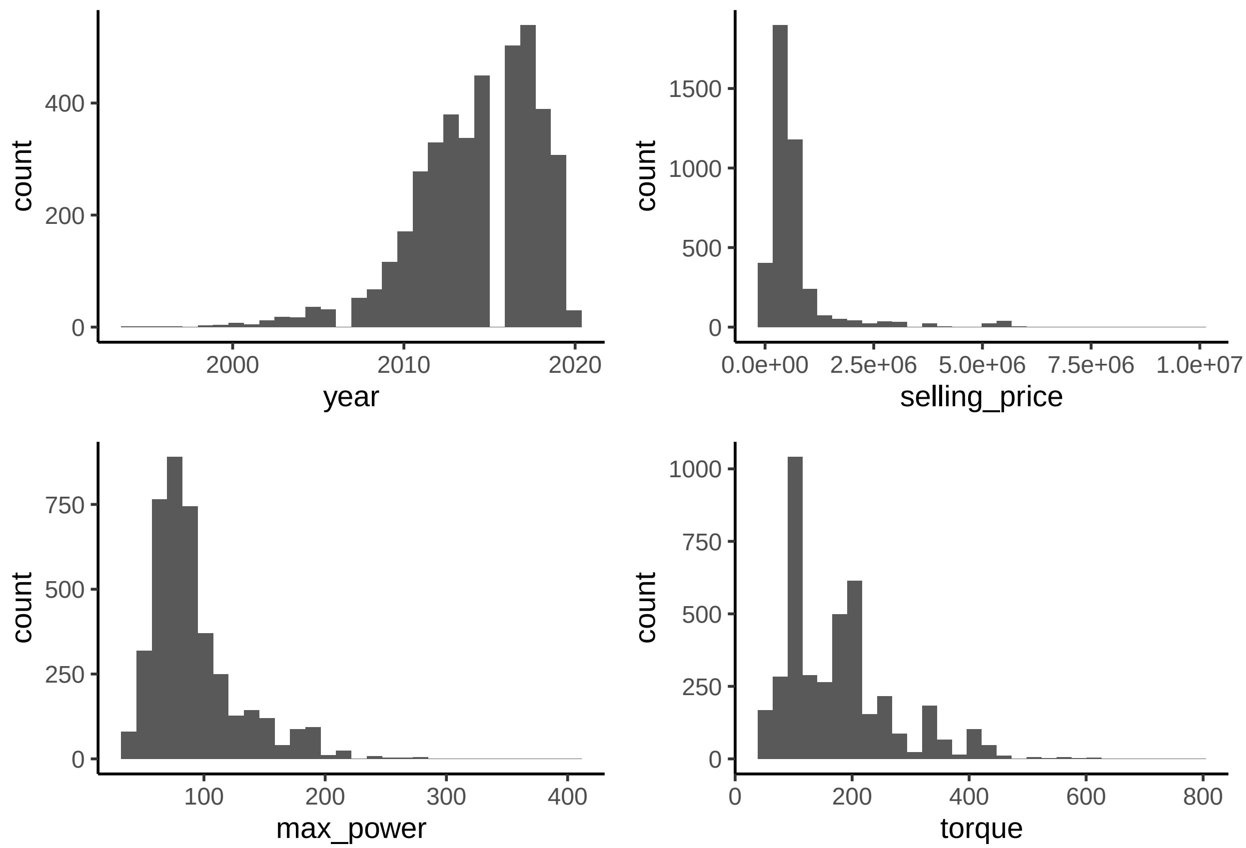
We remove data after visualizing and split the data then.

```r
# set.seed(156)
#
# train_inds <- sample(1:nrow(data), floor(0.8*nrow(data)))
# train_set <- data[train_inds, ]
# test_set <- data[-train_inds, ]
#
# cat('Size of training set: ', nrow(train_set), '\n',
#                                    'Size of Test Set: ', nrow(test_set))
```

1.3 Visualize the Data

```r
g1 <- ggplot(data=data) +
  geom_histogram(aes(x=year))

g2 <- ggplot(data=data) +
  geom_histogram(aes(x=selling_price))

g3 <- ggplot(data=data) +
  geom_histogram(aes(x=max_power))

g4 <- ggplot(data=data) +
  geom_histogram(aes(x=torque))

grid.arrange(g1,g2,g3,g4, ncol=2)
```

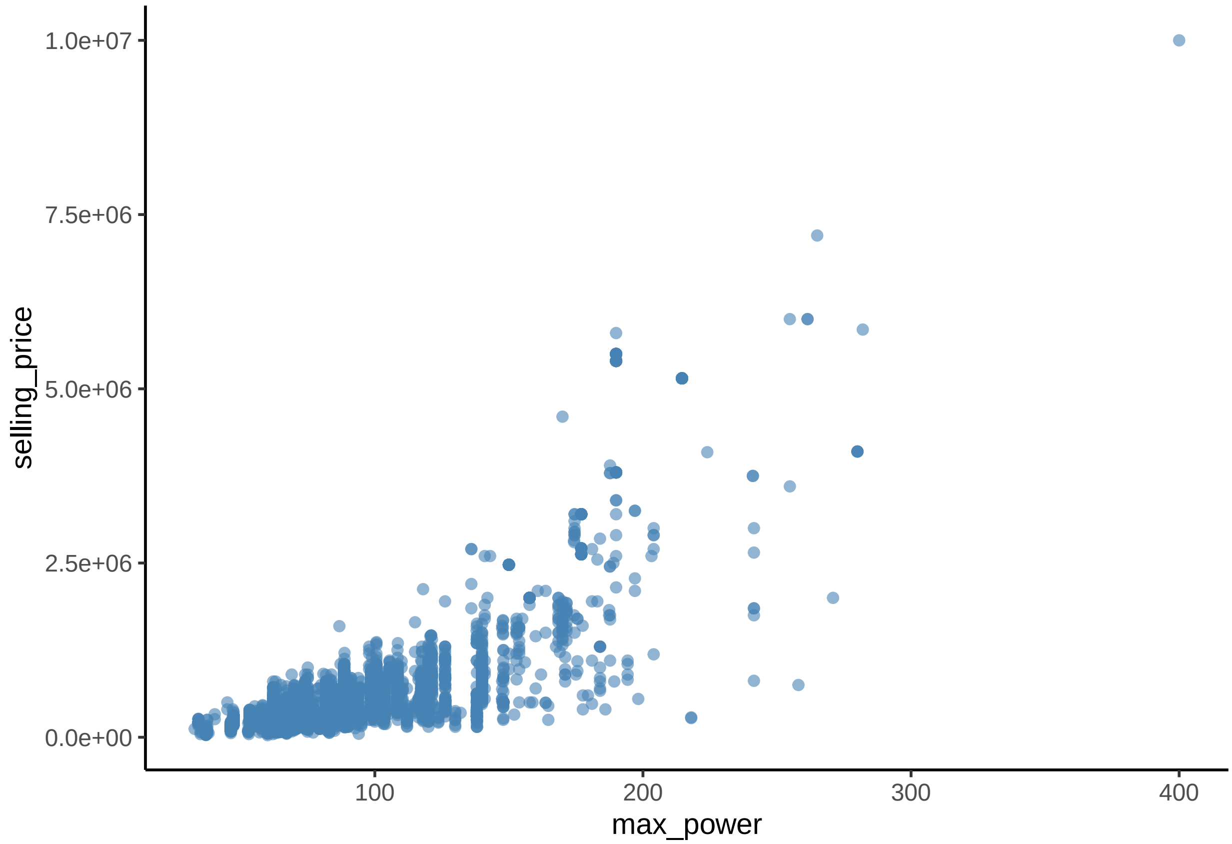
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Both the *max_power* and *torque* distributions appear significantly right skewed. The vast majority of the cars being sold have *max_power* less than 250 bhp, and *torque* less than 450 Nm. The *year* distribution appears significantly left skewed. *selling price*, the dependant variable, appears dramatically right skewed.

Plotting *selling_price* vs. *max_power*

```
ggplot(data=data) +
  geom_point(aes(y=selling_price, x=max_power), colour='steelblue', alpha=0.6)
```



There plot above appears to indicate that there is a positive, exponential correlation between vehicle *max_power* and its *selling_price* on this used car market place. As max_power (brake horse power) of a car increases, its selling_price appears to increases at an increasing rate.

The vast majority of the cars in this sample have *max_power* less than 300 bhp. We will remove what appears to be the one data point above this limit to aid in preventing any overfitting issues. With that data point removed, we will than re-split the data into an 80/20 train/test split sets.

```
data <- data %>%
  subset(max_power<=300)
```

```
set.seed(156)
train_inds <- sample(1:nrow(data), floor(0.8*nrow(data)))
train_set <- data[train_inds, ]
test_set <- data[-train_inds, ]

cat('New Size of Train Set: ', nrow(train_set), '\n')
```

```
## New Size of Train Set:  3269
```

```
cat('New Size of Test Set: ', nrow(test_set))
```

```
## New Size of Test Set:  818
```

Our Train test size shrank by one after sub-setting our data to vehicles with *max_power* <= 300 bhp.

1.4 Fit 4 Models

```r
# 1. Simple Linear Regression Model: selling price vs. max_power
slr <- lm(data=data, selling_price ~ max_power)

# 2. Multiple Linear Regression Model using all variables
mlr <- lm(data=data, selling_price ~ .)

# 3. Polynomial regression: selling_price vs. poly(max_power, p=4)
plr <- lm(data=data, selling_price ~ poly(max_power, 4))

#4. Locally Weighted Regression: selling_price vs. max_power
wlr <- loess(data=data, selling_price ~ max_power)
```

Compare model performance:

```r
set.seed(156)
# Calculate each of the 4 model's RMSE on the train set
slr.preds.train <- predict(slr, newdata = train_set)
rmse.slr.train <- RMSE(slr.preds.train, train_set$selling_price)
mlr.preds.train <- predict(mlr, newdata = train_set)
rmse.mlr.train <- RMSE(mlr.preds.train, train_set$selling_price)
plr.preds.train <- predict(plr, newdata = train_set)
rmse.plr.train <- RMSE(plr.preds.train, train_set$selling_price)
wlr.preds.train <- predict(wlr, newdata = train_set)
rmse.wlr.train <- RMSE(wlr.preds.train, train_set$selling_price)
model.types <- c('simple LM', 'Multiple LM', 'Polynomail Model P=4', 'Weighted LM')
train.rmse <- c(rmse.slr.train, rmse.mlr.train, rmse.plr.train, rmse.wlr.train)

# Calculate each of the 4 model's RMSE on the train set
slr.preds.test <- predict(slr, newdata = test_set)
rmse.slr.test <- RMSE(slr.preds.test, test_set$selling_price)
mlr.preds.test <- predict(mlr, newdata = test_set)
rmse.mlr.test <- RMSE(mlr.preds.test, test_set$selling_price)
plr.preds.test <- predict(plr, newdata = test_set)
rmse.plr.test <- RMSE(plr.preds.test, test_set$selling_price)
wlr.preds.test <- predict(wlr, newdata = test_set)
rmse.wlr.test <- RMSE(wlr.preds.test, test_set$selling_price)

test.rmse <- c(rmse.slr.test, rmse.mlr.test, rmse.plr.test, rmse.wlr.test)
data.frame(model = model.types, train_rmse = train.rmse, test_rmse = test.rmse)
```

```
##                  model train_rmse test_rmse
## 1            simple LM   544269.6  549745.8
## 2          Multiple LM   481590.8  488480.5
## 3 Polynomail Model P=4   454568.8  450839.2
## 4          Weighted LM   479806.3  474527.3
```

When the 4 models were tested on the **train_set**, the models' performance, from best to worst are as follows:

1. Polynomial Regression (p=4)
2. Weighted Linear Regression
3. Multiple Linear Model
4. Simple Linear Regression

When the 4 models were tested on the **test_set_**, it appears as though the model performance does match the same rankings as when they were tested on the train_set, although the invididual model performance does vary. The model performance ranking order is as follows:

1. Polynomial Regression (p=4)
2. Multiple Linear Regression
3. Weighted Linear Model
4. Simple Linear Regression

Even though the ranking of the test and train errors do not appear to not have changed significantly in this instance, it is important to still recognize the train-test split bias. There is a trade off in the train-split method where depending on the nature of the randomness of the data split, in this case using **seed.set(156)**. If a different seed was used, there would be a different set of data in the train and test sets, and therefore it can not be expected to yield the same results / ranking order if a different random seed was used. This is where *cross-validation* comes in handy.

1.5 Cross Validation

For this section of the assignment, we will use k-fold cross validation with K = 10

```
set.seed(156)
ctrl <- trainControl(method='cv', number=10)

# 1. Simple Linear Regression Model: selling price vs. max_power
slr.cv <- train(data=train_set, selling_price ~ max_power, method='lm', trControl=ctrl)

# 2. Multiple Linear Regression Model using all variables
mlr.cv <- train(data=train_set, selling_price ~ ., method='lm', trControl=ctrl)

# 3. Polynomial regression: selling_price vs. poly(max_power, p=4)
plr.cv <- train(data=train_set, selling_price ~ poly(max_power, 4), method='lm', trControl=ctrl)

#4. Locally Weighted Regression: selling_price vs. max_power
wlr.cv <- train(data=train_set, selling_price ~ max_power, method='gamLoess', trControl=ctrl)
```

```
## Loading required package: gam

## Loading required package: splines

## Loading required package: foreach

##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when

## Loaded gam 1.20

## Warning in gam.lo(data[["lo(max_power, span = 0.5, degree = 1)"]], z, w, : eval
## 282

## Warning in gam.lo(data[["lo(max_power, span = 0.5, degree = 1)"]], z, w, :
## upperlimit 281.24

## Warning in gam.lo(data[["lo(max_power, span = 0.5, degree = 1)"]], z, w, :
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(max_power, span = 0.5, degree = 1)"]], z, w, : eval
## 32.8

## Warning in gam.lo(data[["lo(max_power, span = 0.5, degree = 1)"]], z, w, :
## lowerlimit 32.961

## Warning in gam.lo(data[["lo(max_power, span = 0.5, degree = 1)"]], z, w, :
## extrapolation not allowed with blending
```

```r
set.seed(156)

#Calculate Predictions on the Test set
slr.cv.test_preds <- predict(slr.cv, newdata = test_set)
mlr.cv.test_preds <- predict(mlr.cv, newdata = test_set)
plr.cv.test_preds <- predict(plr.cv, newdata = test_set)
wlr.cv.test_preds <- predict(wlr.cv, newdata = test_set)

#Calculate Test Error

slr.cv.test_error <- RMSE(slr.cv.test_preds, test_set$selling_price)
mlr.cv.test_error <- RMSE(mlr.cv.test_preds, test_set$selling_price)
plr.cv.test_error <- RMSE(plr.cv.test_preds, test_set$selling_price)
wlr.cv.test_error <- RMSE(wlr.cv.test_preds, test_set$selling_price)

model.types <- c('simple LM', 'Multiple LM', 'Polynomail Model P=4', 'Weighted LM')
rmses <- c(slr.cv.test_error, mlr.cv.test_error, plr.cv.test_error, wlr.cv.test_error)

data.frame(models = model.types, RMSE = rmses)
```

```
##                  models     RMSE
## 1             simple LM 549813.7
## 2           Multiple LM 489826.8
## 3 Polynomail Model P=4 452535.2
## 4           Weighted LM 455407.8
```

The order of the models' **performance did change** when trained using cross validation when compared to the models tested on the train_set and test_set. Each model also achieved an improvement of their respective performances. The new ranking order of model performance for the models fitted using cross-validation, from best to worst:

1. Weighted Linear Regression
2. Polynomial Regression (P=4)
3. Multiple Linear Regression
4. Simple Linear Regression

Cross-Validation methods randomly split the train_set into partitions, in this case K=10 therefore there are 10 partitions. 9 of these are used as a training set, and the remaining 1 is used as a test/validation set. This is then repeated until all 10 partitions have served as a test set. Cross-Validation is expected to remove much of the traint-test split bias by utilizing all data points for testing and training at one point or another.

1.6 Shrinkage

Fit Models Using Ridge and Lasso Regression

Model Polynomial and Multiple Regressions using L1 and L2 Regularization. Utilizing cv.glmnet() initially to cross-validate the models to find the optimal parameter value for lambda.

```
set.seed(156)

x.train.multiple <- model.matrix(selling_price ~., train_set)[,-1]
x.train.poly <- model.matrix(selling_price ~ poly(max_power,4), train_set)[,-1]
y.train <- train_set$selling_price

x.test.multiple <- model.matrix(selling_price ~., test_set)[,-1]
x.test.poly <- model.matrix(selling_price ~ poly(max_power,4), test_set)[,-1]
y.test <- test_set$selling_price

# Fit multiple and polynomial models on the train set Using Ridge Regression
rfm <- cv.glmnet(x=x.train.multiple, y=y.train, alpha=0, nfolds=10)
rfp <- cv.glmnet(x=x.train.poly, y=y.train, alpha=0, nfolds=10)

# Fit multiple and polynomial models on the train set Using Lasso Regression
lfm <- cv.glmnet(x=x.train.multiple, y=y.train, alpha=1, nfolds=10)
lfp <- cv.glmnet(x=x.train.poly, y=y.train, alpha=1, nfolds=10)

# rfm = ridge fit multiple regression, lfm = lasso fit multiple regression
# rfp = ridge fit polynomial regression, lfp = lasso fit polynomial
```

Determine Optimal Lambda Parameters for each of the above models:

```
rfm$lambda.min
```

```
## [1] 64376.71
```

```
rfp$lambda.min
```

```
## [1] 64376.71
```

```
lfm$lambda.min
```

```
## [1] 10738.68
```

```
lfp$lambda.min
```

```
## [1] 5599.158
```

Re-model the above Ridge and Lasso Regression models with their respective values of lambda shown above.

```
rfm <- glmnet(x=x.train.multiple, y=y.train, alpha=0, lambda=64376.23)
rfp <- glmnet(x=x.train.poly, y=y.train, alpha=0, lambda=64376.23)

lfm <- glmnet(x=x.train.multiple, y=y.train, alpha=1, lambda=9784.615)
lfp <- glmnet(x=x.train.poly, y=y.train, alpha=1, lambda=2423.724)
```

Compare RMSE from Ridge and Lasso Regressions on the test set

```
set.seed(156)

#Calculate RMSE Loss on Test Sets
rfm_preds <- predict(rfm, newx=x.test.multiple)
rfm_rmse <- RMSE(rfm_preds, test_set$selling_price)

rfp_preds <- predict(rfp, newx=x.test.poly)
rfp_rmse <- RMSE(rfp_preds, test_set$selling_price)

lfm_preds <- predict(lfm, newx=x.test.multiple)
lfm_rmse <- RMSE(lfm_preds, test_set$selling_price)

lfp_preds <- predict(lfp, newx = x.test.poly)
lfp_rmse <- RMSE(lfp_preds, test_set$selling_price)

models <- c('multiple', 'poly')
ridge.rmse <- c(rfm_rmse, rfp_rmse)
lasso.rmse <- c(lfm_rmse, lfp_rmse)
data.frame(models = models, Ridge_RMSE = ridge.rmse, Lasso_RMSE = lasso.rmse)
```

```
##     models Ridge_RMSE Lasso_RMSE
## 1 multiple   493201.4   489959.9
## 2     poly   764975.4   841532.8
```

While having trouble modeling a simple regression (1-predictor) using Ridge and Lasso Regression, of the remaining models, the ranking order from best to worst (lowest to highest RMSE) is:

1. **Multiple Regressing** using the **Lasso Regression cost function**
2. **Multiple Regression** using the **Ridge Regression cost function**
3. **Polynomial Regression** using the **Ridge Regression cost function**
4. **Polynomial Regression** using the **Lasso Regression cost function**

It appears as though Polynomial regression using a power of 4 on the *max_power* predictor variable typically performs better than a multiple linear regression model using all the available input variables.

When comparing Lasso vs. Ridge Regression, it appears as though, for a particular model, Lasso Regression outperforms Ridge Regression. Ridge regression's algorithm is set up in a way that all the used predictor variables must stay in the model (i.e. their coefficients may never reach zero), whereas Lasso Regression's constrains on linear regression allows certain variable's coefficients to reach zero.