# 260 Final Project: Boston House Price Prediction

Vivian Shao

2022-12-06

## Introduction

The goal of this project is to answer two key questions. First, how successfully can I apply the machine learning methods I've learned in this course on input features to predict the price of houses in Boston? Second, how do the performances of linear regressions and random forests compare on the same dataset? The dataset was from a research paper titled "Hedonic prices and the demand for clean air" which was authored by David Harrison and Daniel L. Rubenfield in 1978.

Although the data was collected in 1978, I believe many of the insights I can find from this model can still be beneficial towards the housing market today. For example, we can look at the coefficients and p-values to identify which aspects of a house are most important in determining a house price. Additionally, Boston is infamous for its expensive properties and understanding what factors influence that can potentially be important.

The dataset has 506 observations of 13 predictors with the output being the median value of owner-occupied homes in $1000s.

### Feature Description

1) CRIM: per capita crime rate by town
2) ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3) INDUS: proportion of non-retail business acres per town
4) CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise)
5) NOX: nitric oxides concentration (parts per 10 million) [parts/10M]
6) RM: average number of rooms per dwelling
7) AGE: proportion of owner-occupied units built prior to 1940
8) DIS: weighted distances to five Boston employment centres
9) RAD: index of accessibility to radial highways
10) TAX: full-value property-tax rate per $10,000[/10k]$
11) PTRATIO: pupil-teacher ratio by town
12) B: The result of the equation B=1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
13) LSTAT: % lower status of the population

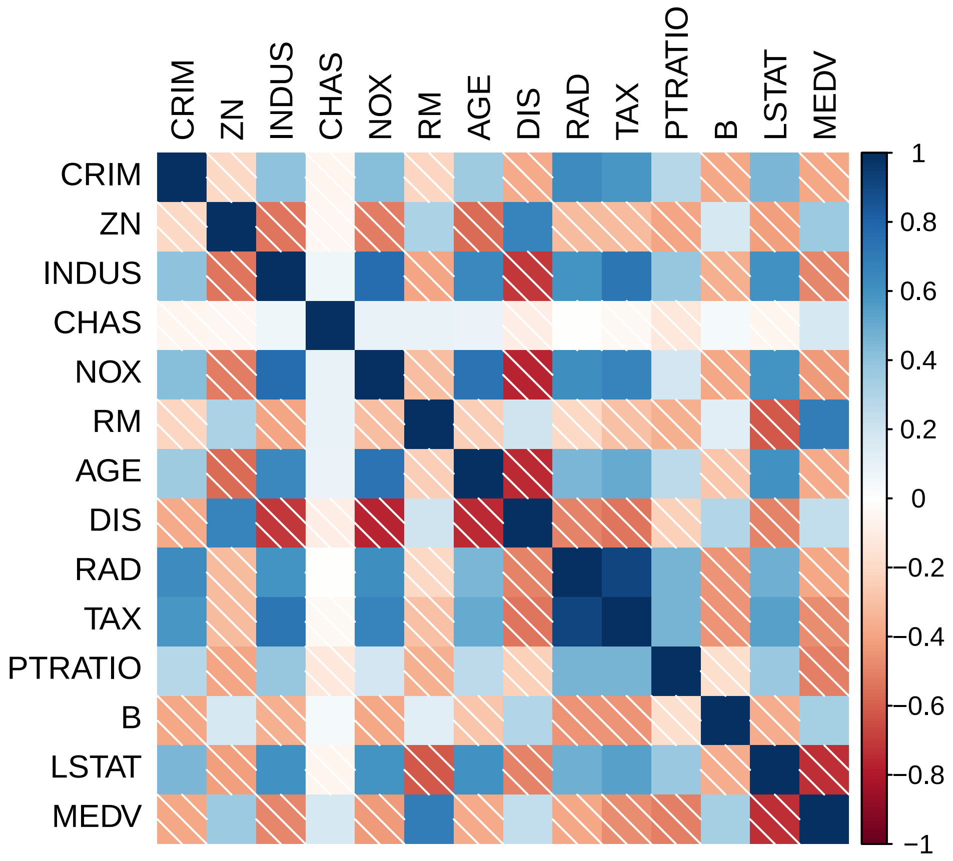
```r
dat <- read.csv("boston.csv")
head(dat)
```

```
##      CRIM ZN INDUS CHAS   NOX    RM  AGE    DIS RAD TAX PTRATIO      B LSTAT
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   MEDV
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

**Exploratory Data Analysis**

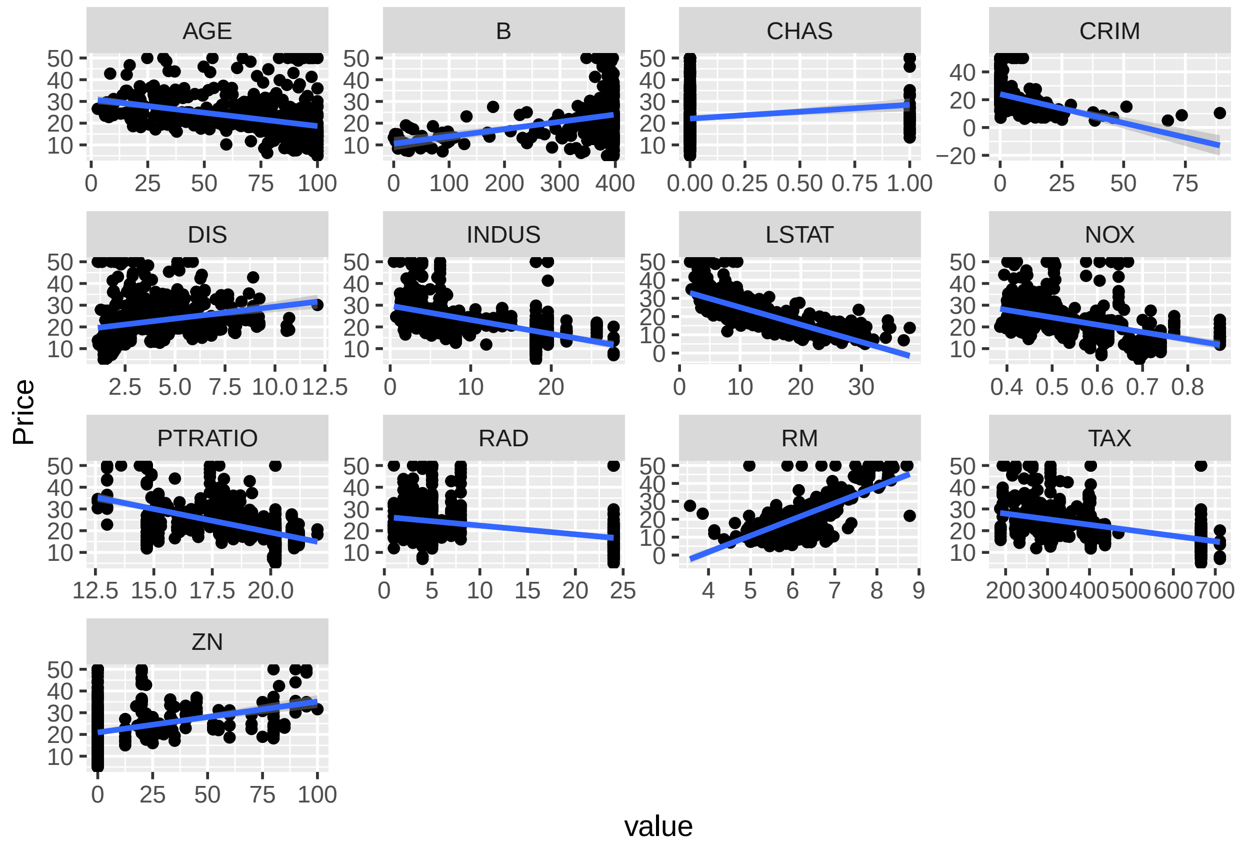
To better understand the dataset and build a prediction model, I performed some exploratory data analysis.

```r
dat <- read.csv("boston.csv")
dat.cor <- cor(dat)
corrplot(dat.cor,
         method = "shade",
         type = "full",
         diag = TRUE,
         tl.col = "black",
         bg = "white",
         col = NULL)
```



In the first plot, which is a correlation matrix, we can see which predictors are most correlated with the desired output. The darker the color, the stronger the correlation. Some significant covariates include RM (average number of rooms per house) and LSTAT (% of the population that's lower status).

```
dat |>
  gather(-MEDV, key = "var", value = "value") |>
  ggplot(aes(x = value, y = MEDV)) +
  geom_point() +
  geom_smooth(method = 'lm') +
  facet_wrap(~ var, scales = "free") +
  ylab('Price')
```



In the second EDA plot, I plotted every variable in a facet plot to better understand the positive, negative, or even insignificant linear relationship each variable has with the house price. Additionally, I can get a sense of the data distribution for each predictor. For example, it's clear from this plot that CHAS (the location in relation to the Charles River) is binary and therefore, should be adjusted for accordingly when building the models.

Based on the largely continuous data and the continuous outcome in the dataset, I decided to use two predictive models we've learned this semester: multivariate linear regression and random forest.

Multivariate linear regressions use multiple independent variables to predict the outcome of a dependent variable. By fitting a regression model to the data, you can use the fitted regression equation to predict the outcome of new observations. On the other hand, random forests are an ensemble learning method that is made up of multiple decision trees. However, a single decision tree, although it helps you arrive at an answer as you traverse the nodes, can be prone to bias or overfitting. When you use multiple independent decision trees in a random forest, it can create more accurate results.

Additionally, I will be using k-fold cross validation to prevent overfitting and to evaluate the performance of my models. K-fold cross-validation is a resampling technique used to evaluate models. The data sample is split into k folds and each portion of the data is used for training, validating, and testing. The final model is then fed the test data to obtain a performance score

For both models, I will also be using the R-squared value to compare prediction accuracy. R-squared indicates how well a regression model is able to predict the responses of new observations. It ranges from of scale of 0 to 1 with 1 indicating the predcitions are identical to the observations.

Overall, I believe these prediction methods and the comparison of a parametric and non-parametric model on the same dataset will be interesting insight.

## Results

In order to perform the k-fold cross validation and also measure how well it is able to create predictions on unseen data, I first split my dataset randomly into 80% that would be fed into the cross-validation and create a cross-validated model. Then, I used the 20% I held out from the cross validation to use as unseen data to test the cross-validated model's predictive performance. Therefore, 404 of the 506 observations were used randomly chosen to be used when cross-validation and the remaining 102 observations were used to make new predictions. I wanted to ensure that I could allocate enough data for my cross-validated model while still having enough "unseen" data for my eventual model evaluation.

Additionally, based on the initial exploratory data analysis where I found that the CHAS variable was binary, I used the as.factor() tranform it before using it for modeling.

```r
set.seed(10)
dat <- read.csv("boston.csv")
split <- sample(1:nrow(dat), floor(0.8*nrow(dat)))

#cv dataset
old <- dat[split,]
old$CHAS <- as.factor(old$CHAS)

#held-out dataset
new <- dat[-split,]
new$CHAS <- as.factor(new$CHAS)
```

To create the held-out dataset which I will use for evaluation, I split the dataset and took away the 14th column to save it as the results column. In doing so, my model will be able to create predictions objectively without influence from the actual MEDV observations. Therefore, when testing the cross-validated model, I only input columns 1 to 13 which were the predictors introduced in the report introduction.

```r
new_true <- new[14]
new_true_vector <- new_true[['MEDV']]
new_dat <- new[1:13]
```

### Linear Regression

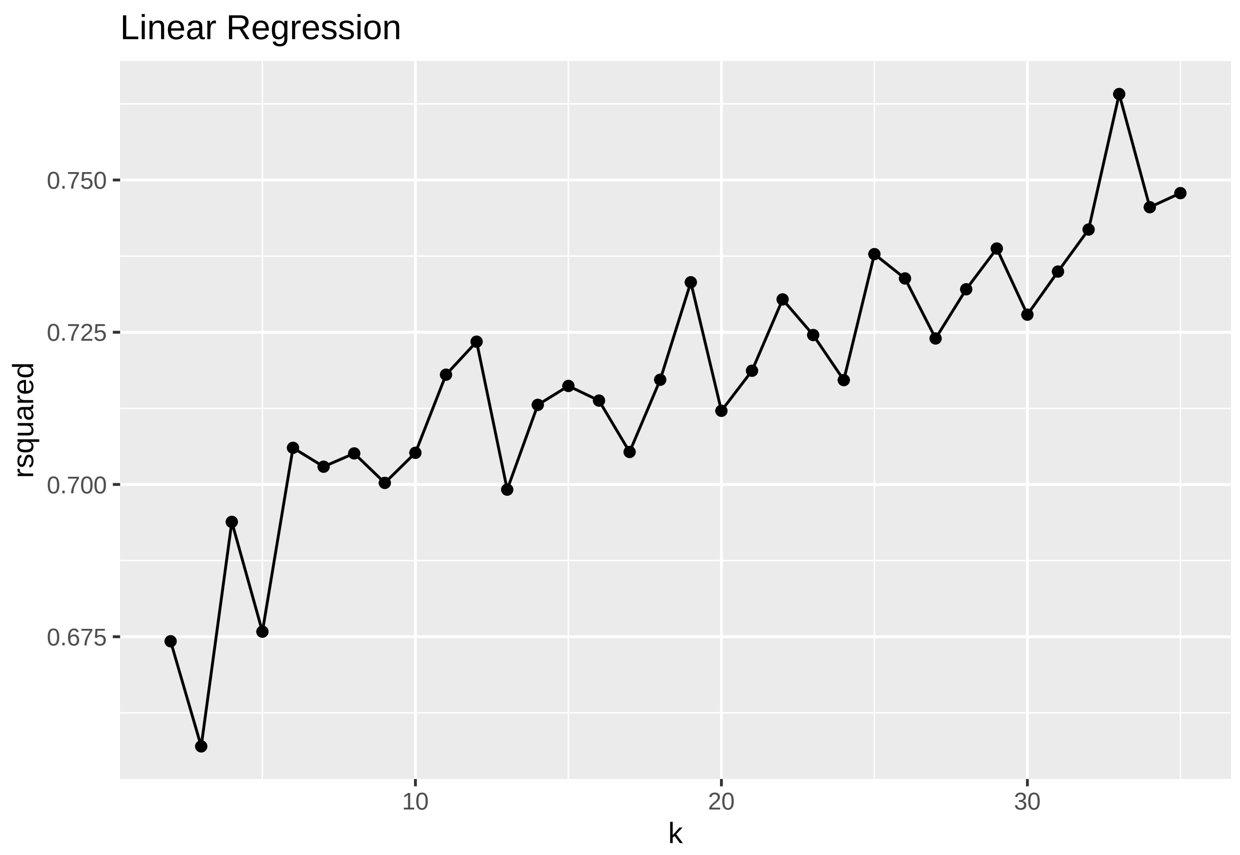
The first model I built was a multivariate linear regression model using k-fold cross validation. To begin, I fit a linear model on the data and analyzed the model summary. It's important to note from this initial model summary which variables are statistically significant and provide the most predictive power. We can see that variables such as LSTAT and RM are extremely statistically significant and these findings also align with our initial data analysis.

```
mod1 <- lm(MEDV ~ ., data = old)
summary(mod1)
```

```
##
## Call:
## lm(formula = MEDV ~ ., data = old)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.6092  -2.7220  -0.5774   1.7397  25.9586
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  38.177696   5.780878   6.604 1.31e-10 ***
## CRIM         -0.077279   0.052417  -1.474 0.141205
## ZN            0.037210   0.015294   2.433 0.015425 *
## INDUS         0.013338   0.073135   0.182 0.855383
## CHAS1         2.750736   1.068721   2.574 0.010425 *
## NOX         -18.381466   4.358511  -4.217 3.08e-05 ***
## RM            3.524380   0.473022   7.451 6.03e-13 ***
## AGE           0.004915   0.014832   0.331 0.740522
## DIS          -1.426138   0.221958  -6.425 3.84e-10 ***
## RAD           0.264717   0.076744   3.449 0.000623 ***
## TAX          -0.011338   0.004333  -2.617 0.009223 **
## PTRATIO      -0.968737   0.151327  -6.402 4.42e-10 ***
## B             0.010705   0.003286   3.257 0.001223 **
## LSTAT        -0.540427   0.058027  -9.313  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.876 on 390 degrees of freedom
## Multiple R-squared:  0.7186, Adjusted R-squared:  0.7092
## F-statistic:  76.6 on 13 and 390 DF,  p-value: < 2.2e-16
```

I then began to build my predictive model using the Caret package and the trainControl() function to specify the number of folds. An important aspect of cross-validation I experimented with was whether the number of folds in cross validation significantly impacted the model's predictive performance. To better understand this, I plotted the overall r-squared value of the cross validated linear regression model based on different numbers of folds.

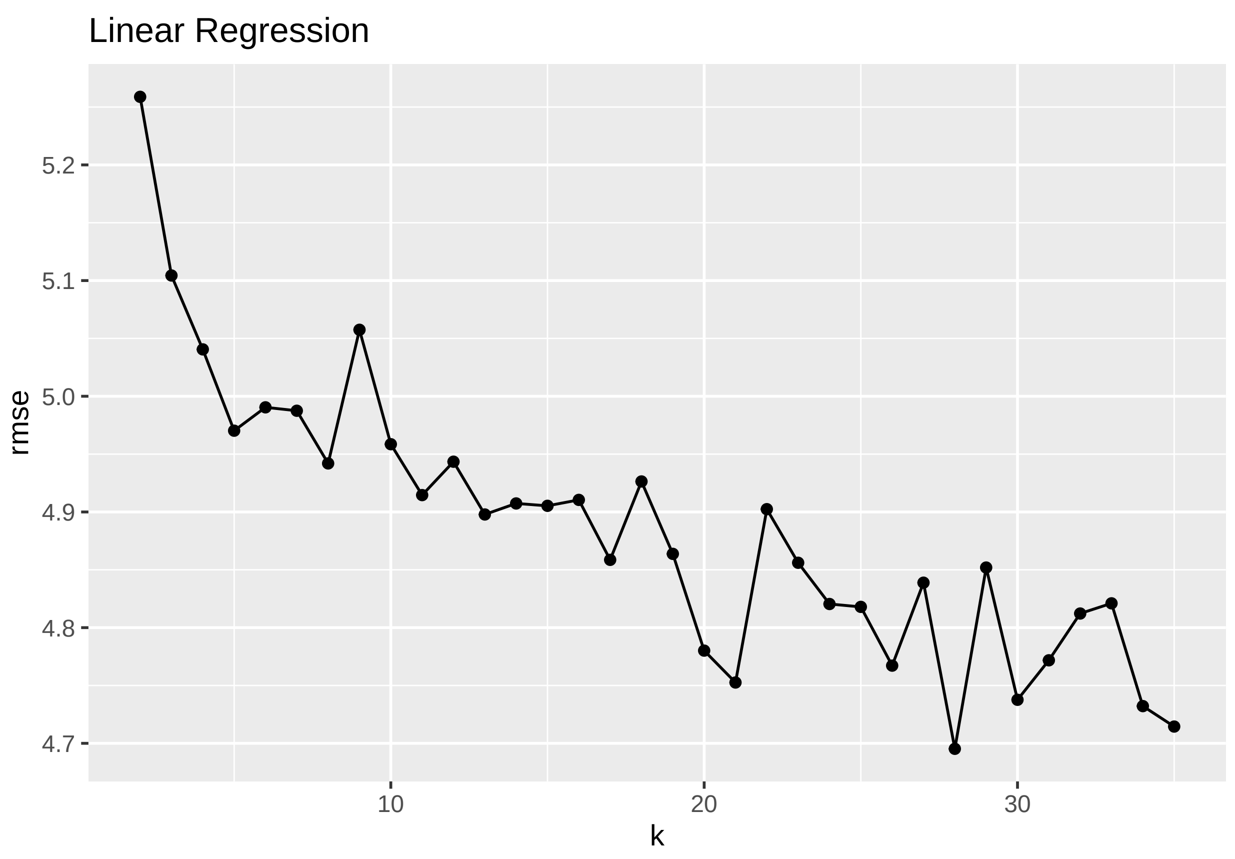
```
columns = c("k", "rsquared")
test <- data.frame(matrix(nrow = 1, ncol = length(columns)))
colnames(test) = columns
for(i in 2:35){
  new <- rep(i, ncol(test))
  test[nrow(test) + 1, ] <- new
  ctrl <- trainControl(method = "cv", number = i)
  model <- train(MEDV ~ ., data = old, method = "lm", trControl = ctrl)
  test$k[i] <- i
  test$rsquared[i] <- model$results[1,3]
}
ggplot(data=test, aes(x=k, y=rsquared)) +
  geom_line()+
  geom_point()+ggtitle("Linear Regression")
```



As we can see from the plot, overall, there's an upwards trend where as k increases, so does the model's overall r-squared value. However, we can see there's also a lot of variation between points. In the end, I chose to use 10-fold cross validation for my linear regression model because its performance seemed consistent.

I actually found this slight increase to be a little unexpected so to double check, I also graphed the RMSE. This verified my findings from above when I used r-squared. As we can see, as k increases, RMSE slightly decreases.

```
columns = c("k", "rmse")
test <- data.frame(matrix(nrow = 1, ncol = length(columns)))
colnames(test) = columns
for(i in 2:35){
  new <- rep(i, ncol(test))
  test[nrow(test) + 1, ] <- new
  ctrl <- trainControl(method = "cv", number = i)
  model <- train(MEDV ~ ., data = old, method = "lm", trControl = ctrl)
  test$k[i] <- i
  test$rmse[i] <- model$results[1,2]
}
ggplot(data=test, aes(x=k, y=rmse)) +
  geom_line()+
  geom_point()+ggtitle("Linear Regression")
```



After creating a cross-validated model using the code below, we can, once again, print the model summary of the overall performance of the cross-validation.

```
ctrl_lm <- trainControl(method = "cv", number = 10)
model_lm <- train(MEDV ~ ., data = old, method = "lm", trControl = ctrl_lm)
print(model_lm)
```

```
## Linear Regression
##
## 404 samples
##  13 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 363, 364, 363, 362, 364, 365, ...
## Resampling results:
##
##   RMSE     Rsquared   MAE
##   4.90471  0.7091684  3.510464
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

The overall r-squared value for this model was 0.709. We can dive even further by looking at the individual metrics of each fold.

```r
print(model_lm$resample)
```

```
##        RMSE  Rsquared      MAE Resample
## 1  5.855003 0.6510302 3.834273   Fold01
## 2  5.281898 0.6494258 3.623808   Fold02
## 3  4.428367 0.7264326 3.259004   Fold03
## 4  7.166912 0.5683772 4.354843   Fold04
## 5  4.217649 0.7594249 3.021822   Fold05
## 6  3.379411 0.7788667 2.754668   Fold06
## 7  3.627867 0.8447056 2.765236   Fold07
## 8  4.287221 0.8339589 3.143411   Fold08
## 9  3.872389 0.8005424 3.221766   Fold09
## 10 6.930388 0.4789201 5.125813   Fold10
```

As we can see, while each fold's performance varies slightly, they average out to about 0.709. We can also see that the RMSEs are also as expected.

The final model's coefficient and intercept estimates after cross validation are:

```r
print(model_lm$finalModel)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Coefficients:
## (Intercept)         CRIM           ZN        INDUS        CHAS1          NOX
##   38.177696    -0.077279     0.037210     0.013338     2.750736   -18.381466
##          RM          AGE          DIS          RAD          TAX      PTRATIO
##    3.524380     0.004915    -1.426138     0.264717    -0.011338    -0.968737
##           B        LSTAT
##    0.010705    -0.540427
```

I then moved on to generating new predictions on unseen data using this cross-validated model using the predict() function. This allows me to calculate the model's r-squared value by comparing the predicted house prices with the actual observed house prices.

```r
new_prediction <- predict(model_lm, newdata=new_dat)
new_prediction <- as.numeric(new_prediction)
new_true <- as.numeric(new_true_vector)

cor(new_true, new_prediction)^2
```

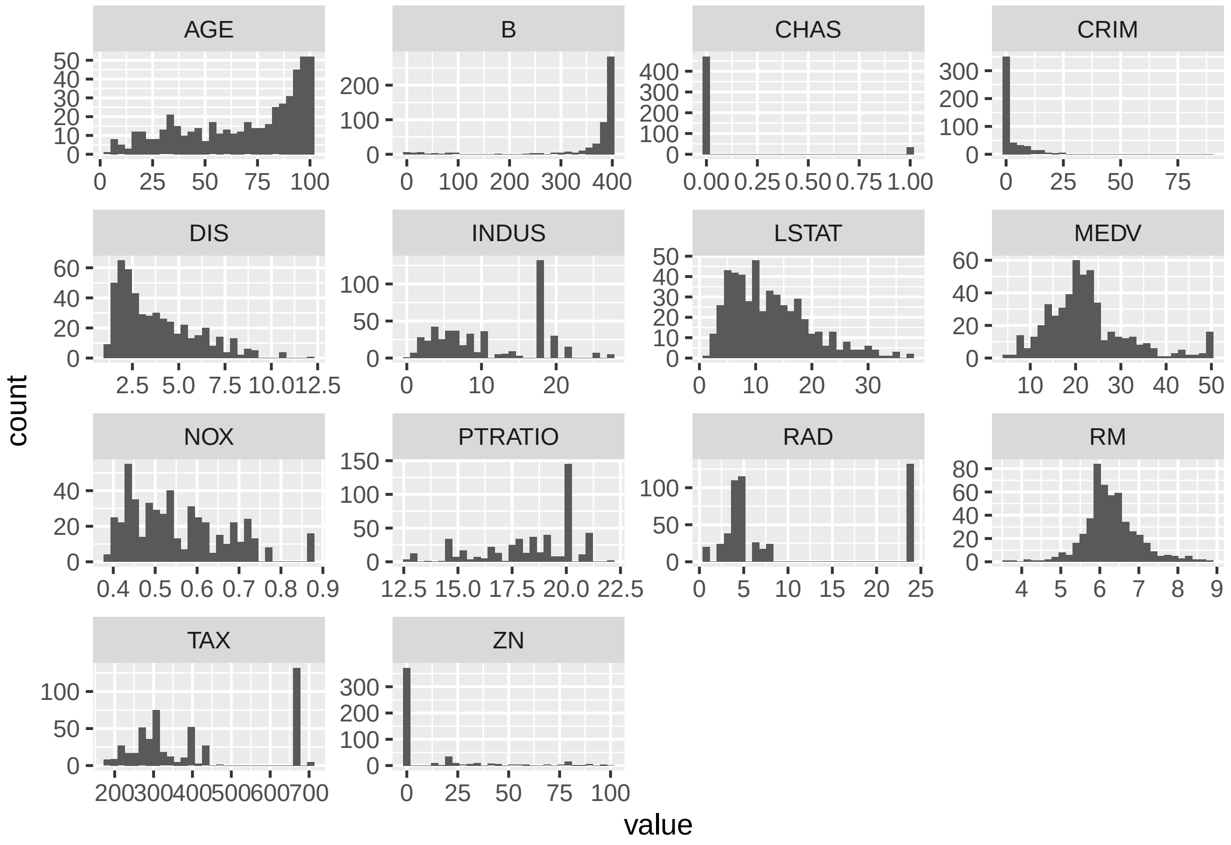
```
## [1] 0.8075232
```

The final r-squared value for the predictions was 0.808 which is relatively close to the r-squared value from the cross-validation.

**Random Forest**

My next method was using a non-parametric model, random forest. Although there are some cases when a linear regression can outperform a random forest such as when the underlying function is truly linear, I believed that random forest would probably outperform the linear regression model in this project based on my prior experience and knowledge. For example, linear models assume that the predictors are linear and independent which is often hard to meet in real-world data. Additionally, the benefit of a random forest is that it's robust to outliers and nonlinear data while also able to handle unbalanced data. Additionally, because random forests are an ensemble machine learning algorithm and averages the decision trees, random forests have low bias and moderate variance.

```r
dat |>
  gather() |>
  ggplot(aes(value)) +
  facet_wrap(~ key, scales = "free") +
  geom_histogram(bins = 30)
```



Here, I plotted the distribution of each covariate and their count. The plot shows that some variables such as RM (number of rooms) are relatively normal distributed. However, other distributions are skewed and such as RAD (accessibility to radial highways).

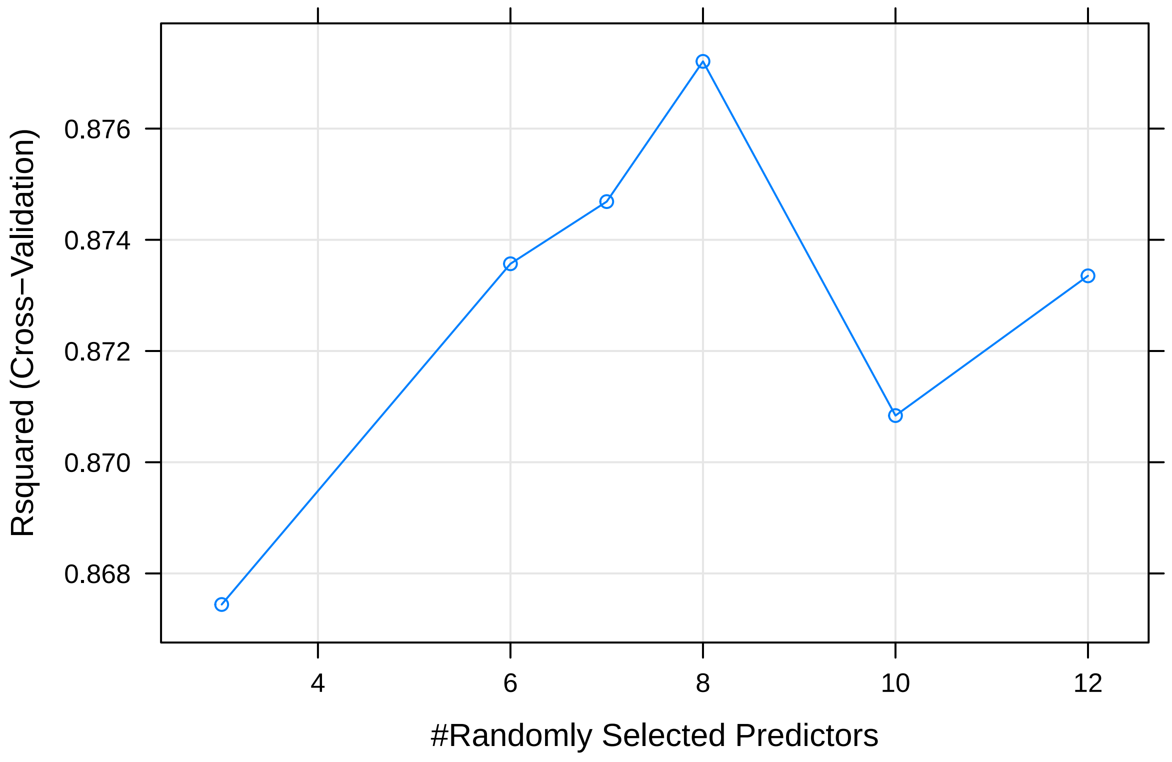
Unlike the cross validation for linear model, the cross validation for random forest actually selects the final model by outputting the most optimal mtry which is the number of variables to randomly sample as candidates at each split. I experimented with the number of folds and saw the best r-squared at 10 folds, although the benefit was pretty small. Additionally, I set the tunelength to be 10 in my train() function to observe how the number of variables randomly sampled as candidates at each split can impact the model's performance.

```
set.seed(10)
control <- trainControl(method='cv',
                        number=10,
                        search = 'random')
rf_random <- train(MEDV ~ .,
                   data = old,
                   method = 'rf',
                   metric = 'Rsquared',
                   tuneLength  = 10,
                   trControl = control)
print(rf_random)
```

```
## Random Forest
##
## 404 samples
##  13 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 364, 363, 363, 363, 363, 364, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    3    3.429923  0.8674413  2.300892
##    6    3.247348  0.8735693  2.222601
##    7    3.244271  0.8746873  2.224622
##    8    3.186096  0.8772092  2.194748
##   10    3.248403  0.8708391  2.228277
##   12    3.204338  0.8733523  2.222294
##
## Rsquared was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 8.
```

We can see from the plot below that the r-squared value increases gradually and peaks at a little above 0.88 when mtry = 8 in this scenario and then dips at mtry = 10. However, I did notice that for other folds in cross-validation, the final value for mtry changes although all performances peak at a little over 0.88 r-squared.

```
plot(rf_random)
```



Therefore, my final model after cross validation uses 8 randomly select predictors. Furthermore, we can look at each fold's performance in the final random forest model. Already, we can see that the random forest model has a slightly better performance than the multivariate linear regression from above as indicated by the lower RMSE and higher r-squared values in each fold.

```
rf_random$resample
```

```
##        RMSE  Rsquared      MAE Resample
## 1  2.539109 0.9182657 1.831180   Fold04
## 2  3.093713 0.9089507 2.041672   Fold09
## 3  3.215980 0.8832607 2.074490   Fold02
## 4  3.130918 0.8406907 2.222489   Fold05
## 5  2.735806 0.9252174 2.169524   Fold03
## 6  3.513056 0.7591351 2.544553   Fold01
## 7  2.297738 0.9583007 1.813947   Fold07
## 8  3.613509 0.8710782 2.282453   Fold10
## 9  4.233935 0.8108380 2.843767   Fold08
## 10 3.487198 0.8963546 2.123409   Fold06
```

Finally, we can use this final cross-validated random forest model on the held-out data to generate new house price predictions. Comparing the true observed values with the predicted values, the model's final r-squared value was 0.923, about 0.11 better than the linear model's r-squared value.

```
set.seed(10)
new_prediction_rf <- predict(rf_random, newdata=new_dat)
```

```r
new_prediction_rf <- as.numeric(new_prediction_rf)
new_true_rf <- as.numeric(new_true_vector)

cor(new_true_rf, new_prediction_rf)^2
```

```
## [1] 0.9228545
```

## Conclusion

I used this project to compare two models, the linear regression and the random forest, and how well I can predict house prices given several different house features.

As mentioned earlier in the introduction, I had two main questions I sought to answer. First, how successfully can I apply the machine learning methods I've learned in this course on input features to predict the price of houses in Boston? Second, how do the performances of linear regressions and random forests compare on the same dataset?

Overall, I was able to obtain satisfactory predictive performance based on the r-squared values at 0.808 for the linear regression and 0.923 for the random forest.

Additionally, we can see from our model comparison that the random forest performed significantly better than the linear regression on the Boston house price data. By cross-validating, we were able to not only prevent overfitting of the data but also select the best final model for predictions.

However, despite the improved performance of the random forest predictive model, we also can see, first-hand, some of the drawbacks when using a non-parametric model. First, the random forest was noticeably slower than the linear regression model when cross-validating and predicting. In this project, the increased time was inconsequential because of the relatively small dataset, but for more robust datasets, I can see this being an issue.

Second, random forests are a black box algorithm so while its complexity compared to the linear model lends itself to better predictions, it is less interpretable than a linear model. We can no longer understand the effects of the coefficients and the final random forest model as well as our final linear model. Therefore, this project has also showed the importance of not blindly choosing the more complex model. In this case, I believe the random forest is the better choice given its improved prediction performance, but for some other projects with datasets that meet the assumptions of a linear model and where linear models can also perform well, you might give up the power of interpretability if you choose to use a random forest instead.

Overall, I believe my analysis was successful because I was able to train two different algorithms and create predictions while also comparing the advantages and disadvantages of both. If I had more time, I would like to explore using other types of models like Naive Bayes or further exploring how I can transform my covariates in the linear model for better fit.

## References

Fedesoriano. (2021, June 1). Boston house prices-advanced regression techniques. Kaggle. Retrieved December 8, 2022, from https://www.kaggle.com/datasets/fedesoriano/the-boston-houseprice-data

## Appendix: All Code for the Project

```r
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(warning = FALSE, message = FALSE)
library(corrplot)
library(ggplot2)
library(tidyr)
library(caret)
dat <- read.csv("boston.csv")
head(dat)
dat <- read.csv("boston.csv")
dat.cor <- cor(dat)
corrplot(dat.cor,
         method = "shade",
         type = "full",
         diag = TRUE,
         tl.col = "black",
         bg = "white",
         col = NULL)
dat |>
  gather(-MEDV, key = "var", value = "value") |>
  ggplot(aes(x = value, y = MEDV)) +
  geom_point() +
  geom_smooth(method = 'lm') +
  facet_wrap(~ var, scales = "free") +
  ylab('Price')
set.seed(10)
dat <- read.csv("boston.csv")
split <- sample(1:nrow(dat), floor(0.8*nrow(dat)))

#cv dataset
old <- dat[split,]
old$CHAS <- as.factor(old$CHAS)

#held-out dataset
new <- dat[-split,]
new$CHAS <- as.factor(new$CHAS)
new_true <- new[14]
new_true_vector <- new_true[['MEDV']]
new_dat <- new[1:13]
mod1 <- lm(MEDV ~ ., data = old)
summary(mod1)
columns = c("k", "rsquared")
test <- data.frame(matrix(nrow = 1, ncol = length(columns)))
colnames(test) = columns
for(i in 2:35){
  new <- rep(i, ncol(test))
  test[nrow(test) + 1, ] <- new
  ctrl <- trainControl(method = "cv", number = i)
  model <- train(MEDV ~ ., data = old, method = "lm", trControl = ctrl)
  test$k[i] <- i
  test$rsquared[i] <- model$results[1,3]
}
ggplot(data=test, aes(x=k, y=rsquared)) +
  geom_line()+
  geom_point()+ggtitle("Linear Regression")
columns = c("k", "rmse")
test <- data.frame(matrix(nrow = 1, ncol = length(columns)))
colnames(test) = columns
for(i in 2:35){
  new <- rep(i, ncol(test))
  test[nrow(test) + 1, ] <- new
  ctrl <- trainControl(method = "cv", number = i)
  model <- train(MEDV ~ ., data = old, method = "lm", trControl = ctrl)
  test$k[i] <- i
  test$rmse[i] <- model$results[1,2]
}
ggplot(data=test, aes(x=k, y=rmse)) +
  geom_line()+
  geom_point()+ggtitle("Linear Regression")
ctrl_lm <- trainControl(method = "cv", number = 10)
model_lm <- train(MEDV ~ ., data = old, method = "lm", trControl = ctrl_lm)
print(model_lm)
print(model_lm$resample)
print(model_lm$finalModel)
new_prediction <- predict(model_lm, newdata=new_dat)
new_prediction <- as.numeric(new_prediction)
new_true <- as.numeric(new_true_vector)

cor(new_true, new_prediction)^2
dat |>
  gather() |>
  ggplot(aes(value)) +
  facet_wrap(~ key, scales = "free") +
  geom_histogram(bins = 30)
set.seed(10)
control <- trainControl(method='cv',
                        number=10,
                        search = 'random')
rf_random <- train(MEDV ~ .,
                   data = old,
                   method = 'rf',
                   metric = 'Rsquared',
                   tuneLength  = 10,
                   trControl = control)
print(rf_random)
plot(rf_random)
rf_random$resample
set.seed(10)
new_prediction_rf <- predict(rf_random, newdata=new_dat)
new_prediction_rf <- as.numeric(new_prediction_rf)
new_true_rf <- as.numeric(new_true_vector)

cor(new_true_rf, new_prediction_rf)^2
```