Kaggle Housing Prices Final Report Nov, 2019

Patrick Reilly, John Cooper, Zeniff Fragoso

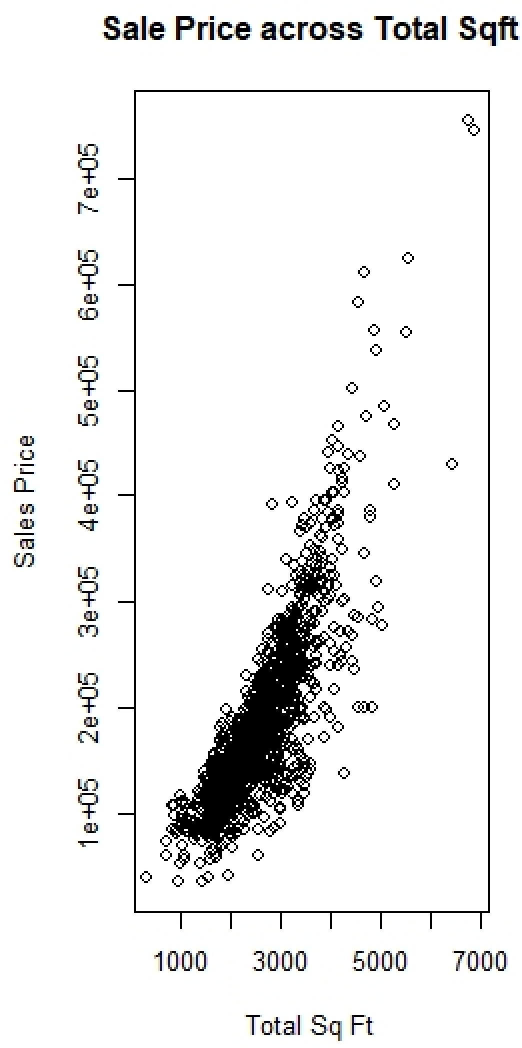
December 6, 2019

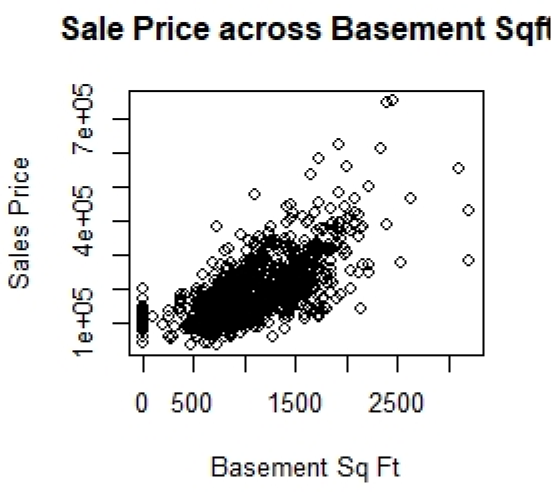
Introduction

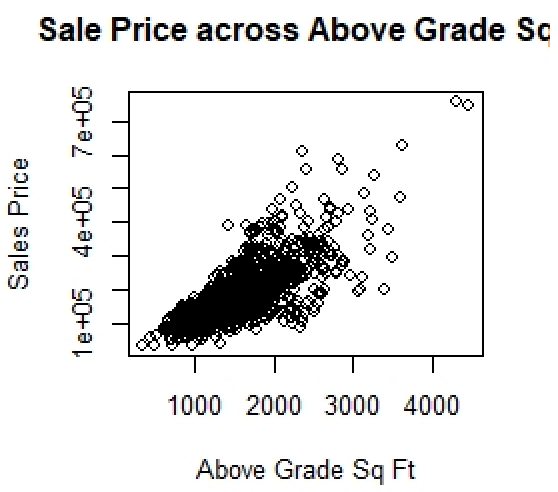
The assigned task is to create a model using R, and our knowledge of statistics and statistical models, to predict the price of homes in Ames, IA. We have been provided with a list of 2,919 properties that have been sold in prior years. The data set includes up to 79 different variables that describe, in a relative fashion, various attributes or characteristics of the associated properties. Our goal is to use this data and the sales prices of 1,460 of the properties (the "Training Set") to build a model to predict the sales price of the remaining 1,459 properties (the "Testing Set") for which we were not provided a sales price. The final accuracy score of our model, based on a RMSLE (Root Mean Squared Log Error), will be provided by Kaggle. We will also be ranked alongside other competitors based on the lowest RMSLE received from the predicted results of our model.

Data Modeling & Cleaning

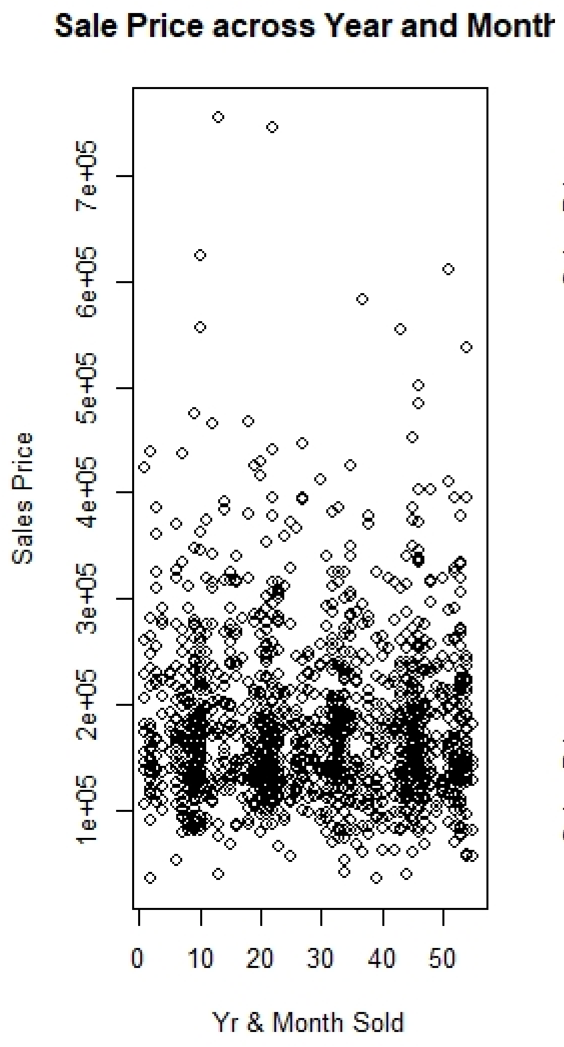
In reviewing the data, a description of the variables, and our limited real estate knowledge, we initially noticed that we needed to add a few variables. The first variable was one for the total square footage of the home. We also combined basement square feet and above grade square feet and called it total_sq_ft. You can see from the scatter plots that the new column named total_sq_ft fits much closer to a linear model than either the total_basement_sf or the above_grade_living_area and made a significant improvement in the performance of our models.

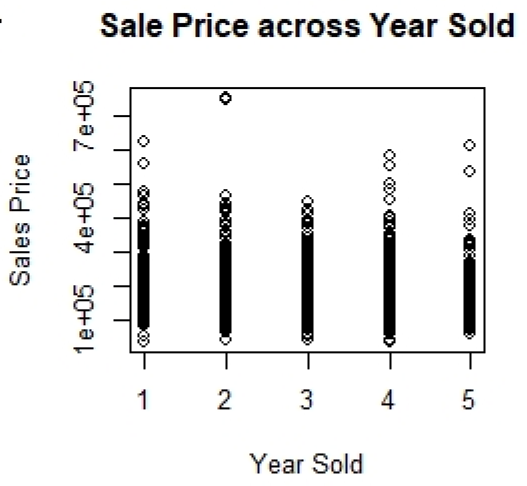




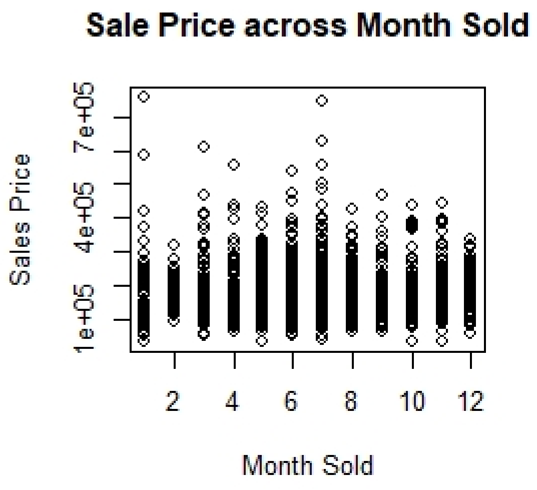


A considerable way into the project We noticed that year and month were two separate factor variables. We started by concatenating the two and left as factors, this made a considerable improvement and made the seasonality of house prices very apparent.





Ultimately, we ended up



changing this to a season factor. Each month of the year was added to its respective season (Spring, Summer, Fall, Winter). This engineered variable ended up providing the highest increase in accuracy to our final model.

Combining Data

We learned pretty early on that we would need to clean the data in the Training Set, and the Test Set. To prevent issues later on, we combined the training data and the test data right off the bat and began looking for issues in the data. The biggest problem that we faced was missing data, or NA’s. With a quick glance, we determined that there were 7,192 instances of missing, or incomplete, data. All that remained was determining how to best deal with these issues. We used our analysis of the NA’s as an opportunity to become more familiar with the intricacies that existed in the data.

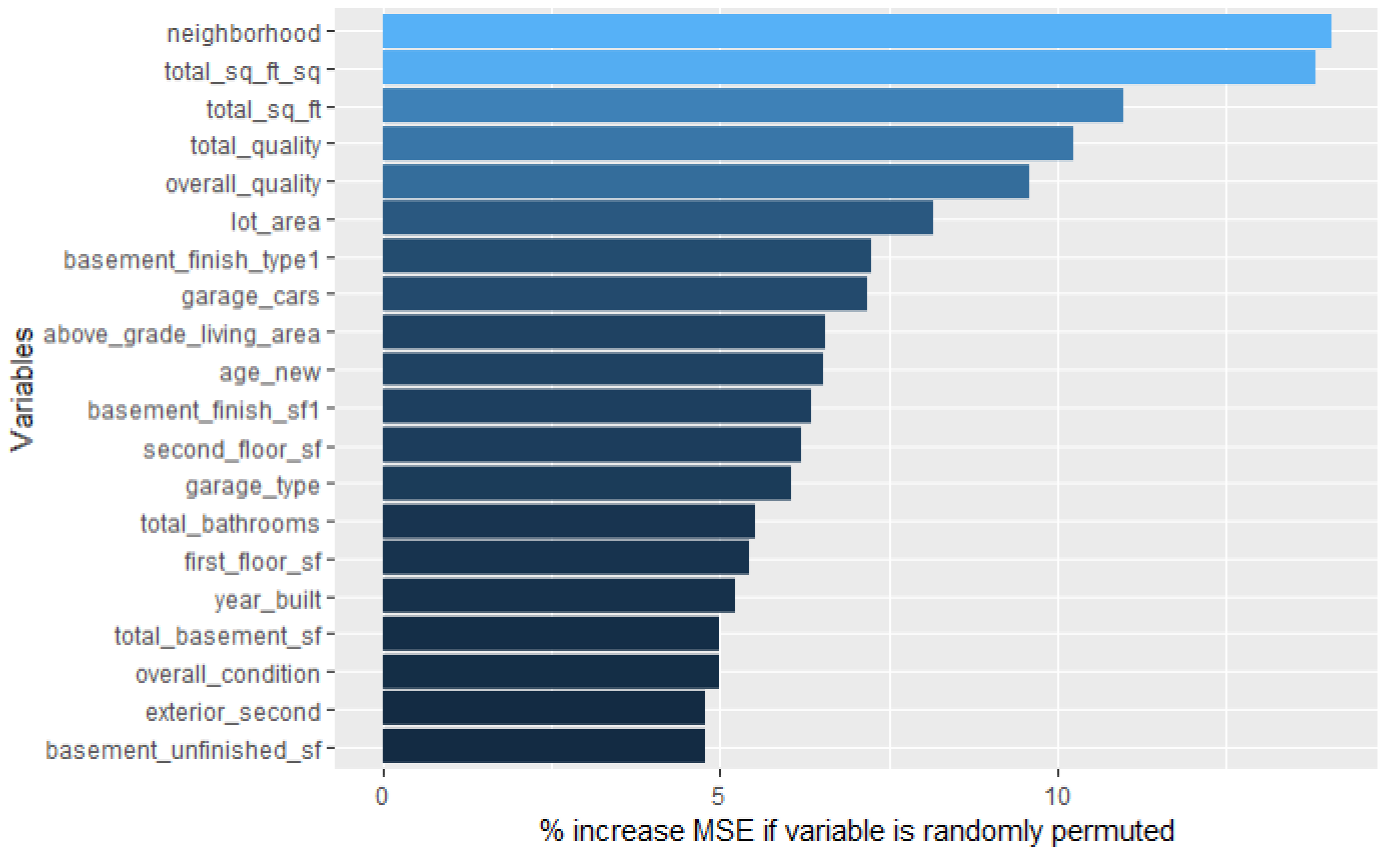
Remembering the different column names was a bit of a challenge early on, so we ended up renaming them to the more familiar “snake case” syntax. Once complete, we continued by looking at the pool_qc column, which was the column with the highest number of missing values in the data. While looking at the pool_qc column we noticed that only 10 properties had data for this variable. In checking the only other pool-related variable, pool_area, we quickly learned that it had 13 results. We were able to infer from this information that the remaining 1,446 properties probably didn’t have a pool. Using some programming magic, we were able to quickly remove many of the missing values with a few lines of code. After reviewing the 3 homes that had pool_area but no pool_qc, we decided that we could use the overall_quality field to help us best predict the likely value of this column by assuming the quality of the property matched the quality of the pool area.

We continued this process as we reviewed the remaining variables with missing data. After a significant amount of time was spent on cleaning and prepping the data, we determined that spending any further time reviewing or updating the missing data would not lead to a significant increase in model accuracy. However, as we neared completion of the project, we had some time to experiment with some of the data and were able to improve our Kaggle score by about .001.

Once the missing data had been dealt with, we decided it would be beneficial to take a closer look at the categorical variables. We found several ordinal variables that we felt would be better represented numerically, and made changes. On the flip side, we also noticed some numerical variables that were better represented categorically, so we changed these variables to factors.

At this point we felt reasonably comfortable that our data was clean. The next step was to determine what variables would be the most important ones in our model. We determined very early in the process that using a log transformation of the sales price, as the outcome variable, more closely resembled a normal distribution. We also did some early testing to verify that the RMSE of our model using a log transformation of sales price resulted in better predictions. Throughout this process, we added several new variables, mostly combinations of other variables, or interactions between multiple variables, to the dataset. Two of the variables we created made the biggest difference in model performance. The first was the combination of the various square footage variables, and the second was the creation of a season variable. The code related to cleaning the data was saved to a separate file. This allowed us the freedom of coding different predictive models independent of changes we might make as we progressed.

Before running the model we had a pretty good idea of which variables would be most important to predicting accurate prices. The graph below illustrates that several of our newly created variables will most likley play an important role in predicting house prices.



Model and Model Development

We started the model development stage of the project by creating a basic ridge regression model. For this model, we used the glmnet method of the caret package in R, while maintaining alpha = 0. This ultimately ended up being updated to a lasso regression model, and later to an elastic-net model. We used the preProcess function of the Caret package to do a BoxCox transformation of the predictor variables to better fit the model. Overall, we were extremely surprised by how well we were able to tune the accuracy of this model. Had we ended the project here, the 0.1212 RMSLE score we received from Kaggle was already a significant improvement of our baseline score from the interim report.

Once we felt we had properly tuned the elastic-net model, we decided to start looking at some more complex models. Next was a random forest regression model. This model took far more computing power to run, which meant longer wait times during cross-validation. We ended up running it several different times using a random search option. As none of the returned models provided an out-of-sample RMSE that was close to the elastic-net model, we ended up leaving this algorithm out of the final model.

The next model we looked at was an extreme gradient boosted tree model. For this analysis, we didn't use the caret package initially. By using the actual xgb package in R, we were able to better fine-tune the hyperparameters of the model to return an out-of-sample RMSE that was much better than even our elastic-net model. This quickly became a front runner for inclusion in our final model, however, there was some concern that it could be overfitting the training data.

We ended up evaluating several other models for inclusion in our final model. These included a support vector machine model (svmRadial, svmLinear), a gradient boosted linear model (glmboost), and an artificial neural network model (ANN). While our understanding of how these models worked was limited initially, we ended up doing additional research to better understand how they worked with the training variables, and how they might be able to support our decision on a final model.

In the end, we went with 5 models. glmboost, glmnet, svmLinear, svmRadial and xgbTree. We then used a off shoot of the caret package called caretEnsemble to run the models and blend the results. This ended up providing us with much more favorable results, as compared to any of the models individually.

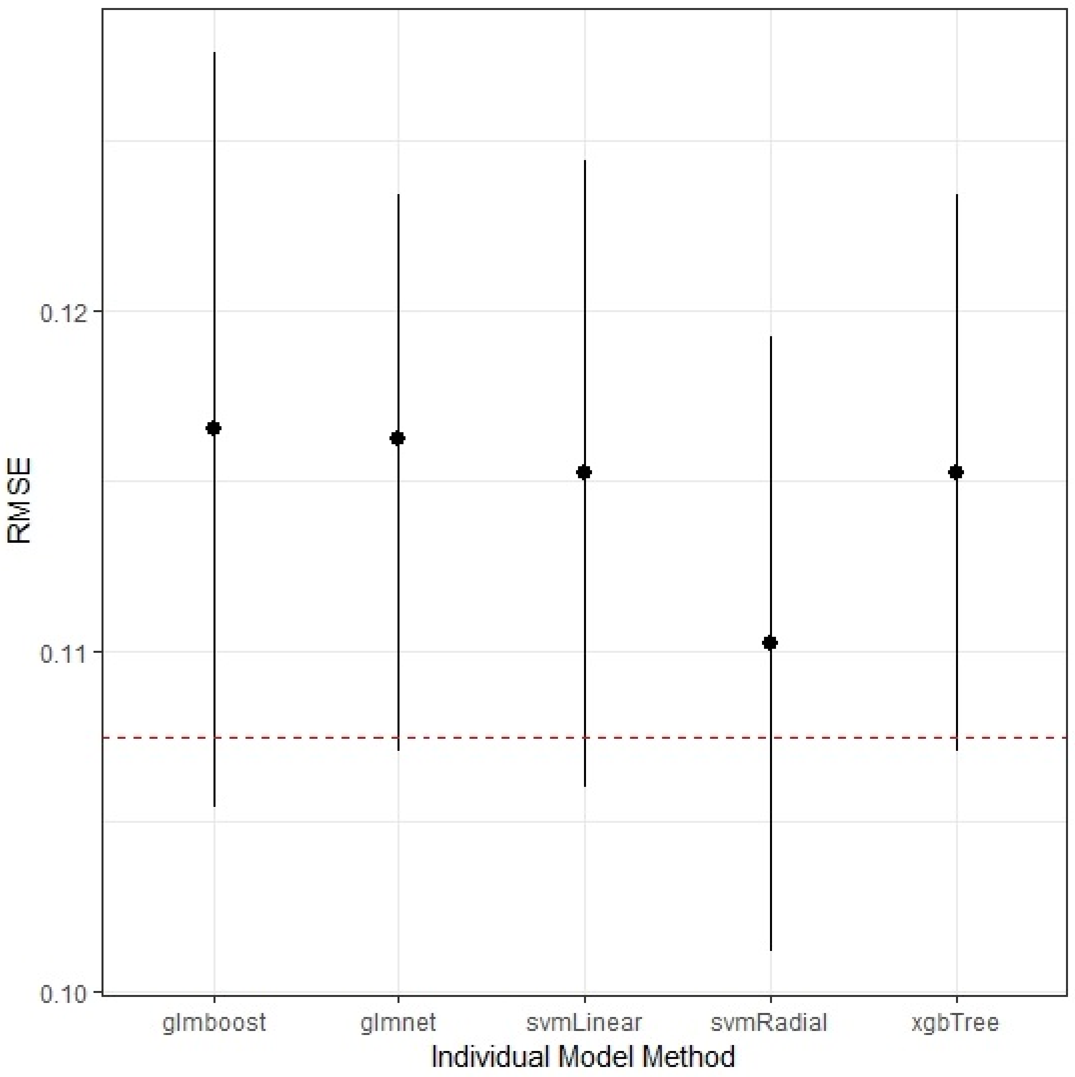
Model Performance

We used separate code files to fine tune the hyperparameters for each model independently. We then combined the models using the caretEnsemble package. This allowed us to run the models together in just a few lines of code, while setting the hyperparameters for each model independently. Then by using greedyEnsemble, which uses a separate linear model to best fit the separate model outcomes (as opposed to just haphazardly combining them), we were able to create a blended model with an in-sample RMSE of 0.112. Cross-validation of our model estimated an out-of-sample RMSE of 0.107. Upon submitting our predicted results to Kaggle, the best score we received was a 0.11427 (rank of 310). Sadly, we only received this Kaggle score once due to variation in the model. However, the model consistently returned a Kaggle score that was on average around 0.11450. We were very pleased with these scores.

parameter <fctr>	RMSE <dbl>	Rsquared <dbl>	MAE <dbl>	RMSESD <dbl>	RsquaredSD <dbl>	MAESD <dbl>
none	0.1071001	0.9283421	0.07299186	0.0150357	0.01780671	0.00686085

1 row

We found it very interesting that the blended model had a noticeably better out-of-sample RMSE than the individual algorithms. The svmRadial model performed the best, compared to the other models, but still resulted in out-of-sample RMSE that was 0.05 higher than the RMSE of the blended model.



Below is a chart showing the top 10 most important variables. We calculated this using dummyVars, and created this list by removing everything after the ".", which removed the value of each variable, and grouped by the original variable, to return a total importance level for the original variables. We can do this because we did not remove variables with near-zero variance ("NZV"). It was interesting to see that each model treated each of the variables differently. For example, above_grade_living_area was overall very important (ranked 4th) however, it was insignificant to the glmboost model (only 0.53) compared to 3.9 for both the svmRadial and svmLinear models. Our goal was to design an aggregated model that used algorithms with a wide variety of approaches, and this table demonstrates that we achieved that goal. Intheory, each of these models combined, should reduce the impact of any overfitting in the individual models.

var <fctr>	sum.overall. <dbl>	sum.glmboost. <dbl>	sum.svmRadial. <dbl>	sum.xgbTree. <dbl>	su
total_sq_ft	9.087399	11.776592	4.9355157	27.052814	
overall_quality	7.674707	6.808377	4.8911809	17.526650	
neighborhood	5.378883	8.080304	4.3476898	1.537112	
above_grade_living_area	3.613621	0.530375	3.9634205	2.000679	
garage_cars	3.201567	2.625667	3.3622911	2.967716	
total_bathrooms	3.133906	2.021632	3.3194268	5.487710	

var <fctr>	sum.overall. <dbl>	sum.glmboost. <dbl>	sum.svmRadial. <dbl>	sum.xgbTree. <dbl>	su
year_built	3.031023	3.608042	2.4982695	2.699262	
first_floor_sf	2.628066	1.283797	2.8444889	1.422050	
kitchen_quality	2.440281	1.198751	3.2541331	2.733103	
sale_condition	2.406342	5.485956	0.8984323	0.833848	

1-10 of 10 rows

The aggregated results were significantly different from the sorted raw output produced by VarImp and provided important insights into the overall importance of specific variables to the outcome variable. Without setting a seed, the results varied each time we ran the model, but total_sq_ft and overall_quality were consistently near the top. Unlike the table above, the results below are the dummy variables that were the most significant.

var <fctr>	overall <dbl>	glmboost <dbl>	svmRadial <dbl>	xgbTree <dbl>	svmLinear <dbl>	glmnet <dbl>
total_sq_ft	8.029904	11.2374855	5.074896	26.980655	5.074896	5.0503499
overall_quality	6.905116	6.3597432	5.029309	15.540021	5.029309	7.2151130
above_grade_living_area	4.157383	0.8175396	4.075348	3.762670	4.075348	5.4664347
garage_cars	3.312246	2.5054694	3.457243	3.297554	3.457243	3.2567829
total_bathrooms	3.177423	1.8016024	3.413168	5.756714	3.413168	1.9943730
year_built	3.009422	3.7064454	2.568821	2.022985	2.568821	4.1291854
first_floor_sf	2.707273	1.1770636	2.924818	1.140946	2.924818	3.3765919
kitchen_quality	2.537316	1.1410211	3.346031	2.119607	3.346031	1.4675358
exterior_quality	2.293225	0.1726431	3.469363	2.276948	3.469363	0.5111536
age	2.233162	1.4644580	2.409343	3.140191	2.409343	1.7092314

1-10 of 10 rows

Summary

Overall, we are very happy with our top reported Kaggle score of 0.11427 and rank of 308 on the Kaggle leaderboard. We learned an incredible amount about R, the caret package, and how to implement at least 9 very different algorithms. This exercise was a very positive experience.