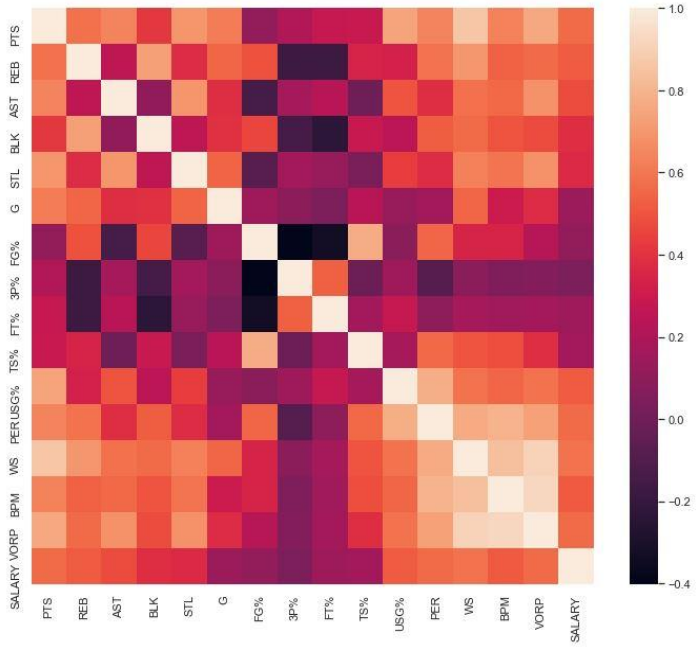
Capstone Final Report

NBA front offices largely determine many of the contracts they give out to players based on how much the market says they are worth. By this I mean teams (and agents) look at what similar players have been paid to determine the worth of the free agent they want to sign. Although this method isn't going anywhere thanks to the relationship between team front offices and agents, teams can at least use past signings in conjunction with past season performances to determine salaries. I used data from NBA.com and from Basketball-Reference.com to create a model that does just that.

I performed an EDA on the player statistics to find relationships amongst the features that may give interesting insights. To help, a heatmap of the correlations between the features was created as seen on the right. The lighter the square that aligns with two features the higher the correlation between those two features are and the vice versa is also true. The white diagonal line just indicates when a feature is being compared to itself. From this heatmap, a lot of relationships can be seen at a high level. There is a very noticeable square of lightness that can be seen on the bottom-right. This square consists of the positive correlations between three of the advanced stats that I included as features. They are the WS (win shares), BPM (box plus-minus), and VORP (value-over-replacement-player). Generally speaking, the better these three stats are the better the player is. In theory, these three stats are meant to show just how good a player is no matter how good their normal box-score stats say. PER (player efficiency rating) and USG% (usage rate) also seem to be part of this group of correlated features, PER more so than USG%. These advanced stats also seem to be positively correlated with PTS (total points scored). This makes total sense: good players tend to score more points.



FT% (free throw percentage) being negatively correlated with FG% (field goal percentage) is interesting. It is understandable to assume players that shoot well from the free throw line shoot well during the game, but this correlation might have an easy explanation. Some of the worst free throw shooters in the league play the center position, the position where most of the points come from within a few feet of the basket. Since a large part of their shots come from so close, they tend to make a large percentage of them hence the high FG%. There are also negative correlations between FG% and both FT% and 3P% (3 pointer percentage). 3 pointers are taken often during games because they are more efficient even though players and

teams as a whole typically make a smaller percentage of 3s than they do 2s. The negative correlation comes from the fact that FG% gives 2s and 3s the same amount of weight, despite the fact that 3s are made less often (and are worth more). It's worth noting that the colorbar for the heatmap only goes as low as -0.4, which means there aren't any strongly negative correlations between any of the features.

To find and develop the best model for this problem a baseline performance had to be established. First, the data was partitioned into training and testing splits at random. The testing split contained 30% percent of the players while the training contained the rest. In order to get this baseline performance, a very simple baseline model was created where its salary prediction for every player in the testing split was the mean salary of the players in the training split. It was not a good model, but I knew that going in. The 4 metrics I used to measure the performance of this model, and all the models I tested, were R^2, Adjusted R^2, MAE, and RMSE. These metrics were applied to the model prediction on the training and testing data. For example, the model was trained on the training data, then the model made predictions on the same data it was trained on and then made predictions on the testing data. As a result, I measured the performance for the mean baseline model using 8 separate metrics. The results can be seen on the table below.

models-*metrics*	*mean R^2 (GridSearchCV)*	*R^2 (train)*	*R^2 (test)*	*Adjusted R^2 (train)*	*Adjusted R^2 (test)*	*MAE (train)*	*MAE (test)*	*RMSE (train)*	*RMSE (test)*
Dummy (mean) Regressor	N/A	0	-0.001144622163	-0.1595744681	-0.4704311638	7067479.768	6951335.557	8290153.791	8507852.202

The “mean R^2 (GridSearchCV)” metric can be ignored for this model since it wasn’t used. At any rate, the R^2 score between the actual training salaries and the predicted training salaries, which I mentioned previously is just the mean of the actual training salaries, is zero. Naturally, this was to be expected. What was also to be expected was the terrible metrics. A below-zero R^2 is very bad and MAE and RMSE scores that high are also bad. The test MAE and test RMSE mean that the mean model was off by about $7 million and $8.5 million, respectively. That is a lot of money to be off by.

Next I tried the sklearn LinearRegression model twice, first with the data as it is and then with the data scaled using the sklearn StandardScaler.

models-*metrics*	*mean R^2 (GridSearchCV)*	*R^2 (train)*	*R^2 (test)*	*Adjusted R^2 (train)*	*Adjusted R^2 (test)*	*MAE (train)*	*MAE (test)*	*RMSE (train)*	*RMSE (test)*
Dummy (mean) Regressor	N/A	0	-0.001144622163	-0.1595744681	-0.4704311638	7067479.768	6951335.557	8290153.791	8507852.202
Linear Regression (unscaled)	N/A	0.6214726672	0.3892925341	0.5610693694	0.1030234095	3993532.992	4959487.834	5100481.887	6644895.294
Linear Regression (scaled)	N/A	0.6214726672	0.3892925341	0.5610693694	0.1030234095	3993532.992	4959487.834	5100481.887	6644895.294

The linear regression models performed much better than the mean regression model across the board. However, scaling the data produced negligible differences or no differences at all for every metric. Despite this I kept scaling the data for the rest of the models I tried since there are wildly different scales for different features.

The next three models I tried were also linear models from sklearn, they were the Ridge model, the Lasso model, and the ElasticNet model.

models-*metrics*	*mean R^2 (GridSearchCV)*	*R^2 (train)*	*R^2 (test)*	*Adjusted R^2 (train)*	*Adjusted R^2 (test)*	*MAE (train)*	*MAE (test)*	*RMSE (train)*	*RMSE (test)*
Dummy (mean) Regressor	N/A	0	-0.001144622163	-0.1595744681	-0.4704311638	7067479.768	6951335.557	8290153.791	8507852.202
Linear Regression (unscaled)	N/A	0.6214726672	0.3892925341	0.5610693694	0.1030234095	3993532.992	4959487.834	5100481.887	6644895.294
Linear Regression (scaled)	N/A	0.6214726672	0.3892925341	0.5610693694	0.1030234095	3993532.992	4959487.834	5100481.887	6644895.294
Ridge Regression (scaled)	0.4327860265	0.5856605149	0.4230847569	0.3422731583	0.1526557367	4209099.691	4715319.277	5336306.281	6458438.588
Lasso Regression (scaled)	0.3745317665	0.6214726262	0.3893765573	0.2747230059	0.1031468186	3993490.105	4958987.241	5100482.162	6644438.164
ElasticNet Regression (scaled)	0.4327837711	0.5784262474	0.4226972406	0.342270543	0.1520865722	4255236.742	4710168.547	5382690.005	6460607.303

These three models require hyperparameters, so to determine the best values for those parameters I used sklearn's GridSearchCV with 5 folds. The ridge model required only one hyperparameter and GridSearchCV determined that the best value was 13.9 which produced a mean R^2 of about 0.433 as can be seen in the chart above. As for the rest of the metrics, it performed much better than the linear model but was very obviously not accurate. The lasso model also required alpha but GridSearchCV determined the best value was 99.9 which was the highest value I allowed. I checked higher value limits but it always chose the highest value but saw a negligible difference in score so I stuck with 99.9. That being said, the lasso didn't perform anywhere near as well as the ridge model but did perform better than the linear models albeit only marginally. The next model I tried was the elastic net model which required alpha along with l1_ratio, another hyperparameter. GridSearchCV determined the best performing combination of alpha and the l1_ratio was 1.6 and 0.9, respectively. This elastic net model had a mean R^2 that was only slightly lower than the score for the ridge model. The same applied to nearly every other metric I used, with the exception being MAE when the testing data was predicted but even then the difference wasn't significant.

The next model I tried was the sklearn DecisionTreeRegressor model. The metrics from using this model can be seen below.

models-*metrics*	*mean R^2 (GridSearchCV)*	*R^2 (train)*	*R^2 (test)*	*Adjusted R^2 (train)*	*Adjusted R^2 (test)*	*MAE (train)*	*MAE (test)*	*RMSE (train)*	*RMSE (test)*
Dummy (mean) Regressor	N/A	0	-0.001144622163	-0.1595744681	-0.4704311638	7067479.768	6951335.557	8290153.791	8507852.202
Linear Regression (unscaled)	N/A	0.6214726672	0.3892925341	0.5610693694	0.1030234095	3993532.992	4959487.834	5100481.887	6644895.294
Linear Regression (scaled)	N/A	0.6214726672	0.3892925341	0.5610693694	0.1030234095	3993532.992	4959487.834	5100481.887	6644895.294
Ridge Regression (scaled)	0.4327860265	0.5856605149	0.4230847569	0.3422731583	0.1526557367	4209099.691	4715319.277	5336306.281	6458438.588
Lasso Regression (scaled)	0.3745317665	0.6214726262	0.3893765573	0.2747230059	0.1031468186	3993490.105	4958987.241	5100482.162	6644438.164
ElasticNet Regression (scaled)	0.4327837711	0.5784262474	0.4226972406	0.342270543	0.1520865722	4255236.742	4710168.547	5382690.005	6460607.303
DecisionTreeRegressor (scaled)	0.3628601174	0.4798729114	0.1060998076	0.2611888595	-0.3129159076	4591332.135	5796544.278	5978845.4	8039259.411

This model requires several hyperparameters, some of which have default values when used. I chose to modify 6 of its hyperparameters, criterion, splitter, max_depth, min_samples_split, min_samples_leaf, and min_weight_fraction_leaf. GridSearchCV determined the best combination of values for this model were criterion = mae, splitter = random, max_depth = 3, min_samples_split = 2, min_samples_leaf = 7, and min_weight_fractions_leaf = 0.0. This model had a mean R^2 worse than the lasso model which

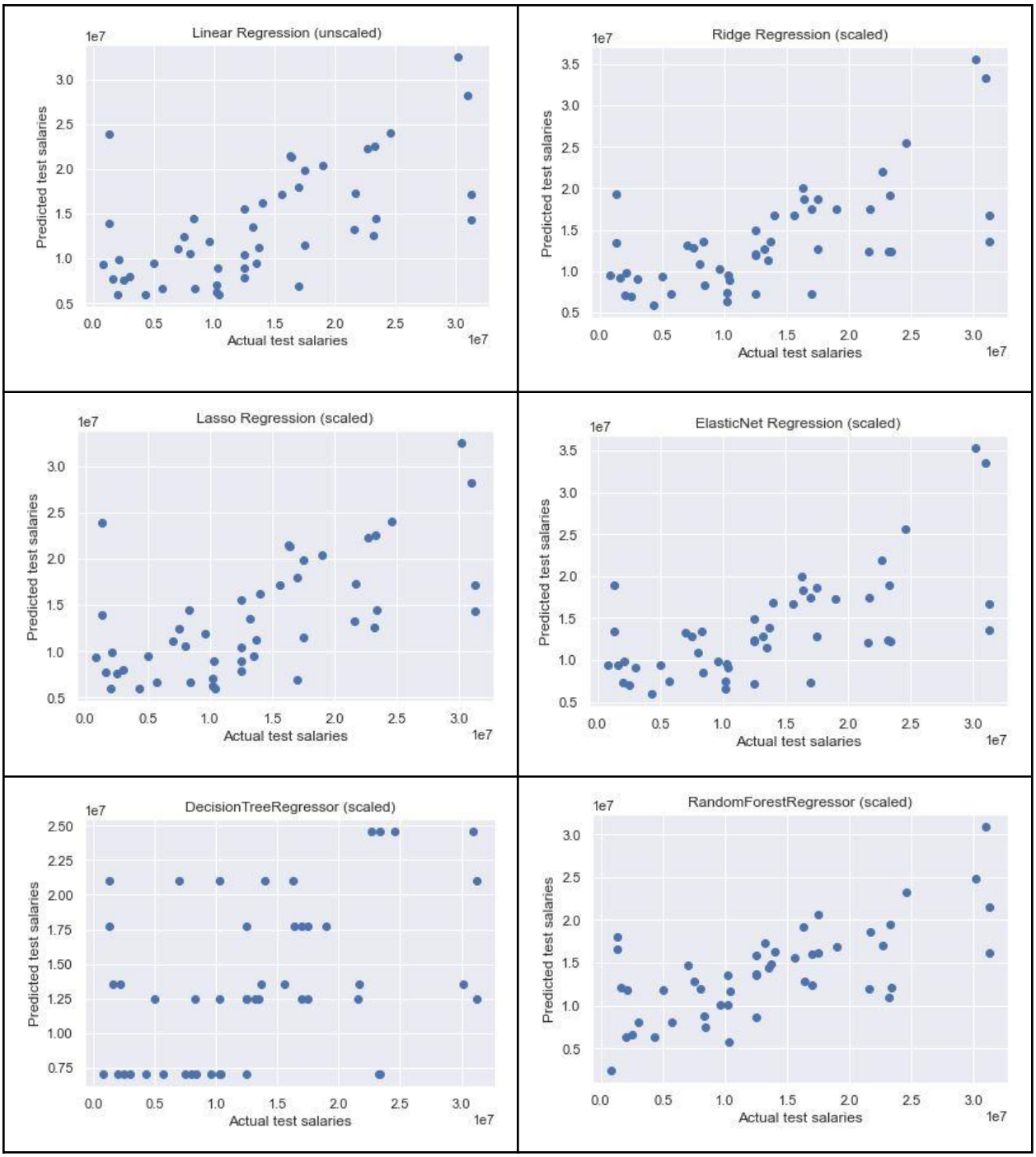
held that title before this model was tried out. Not only that, but with the other metrics it performed much worse.

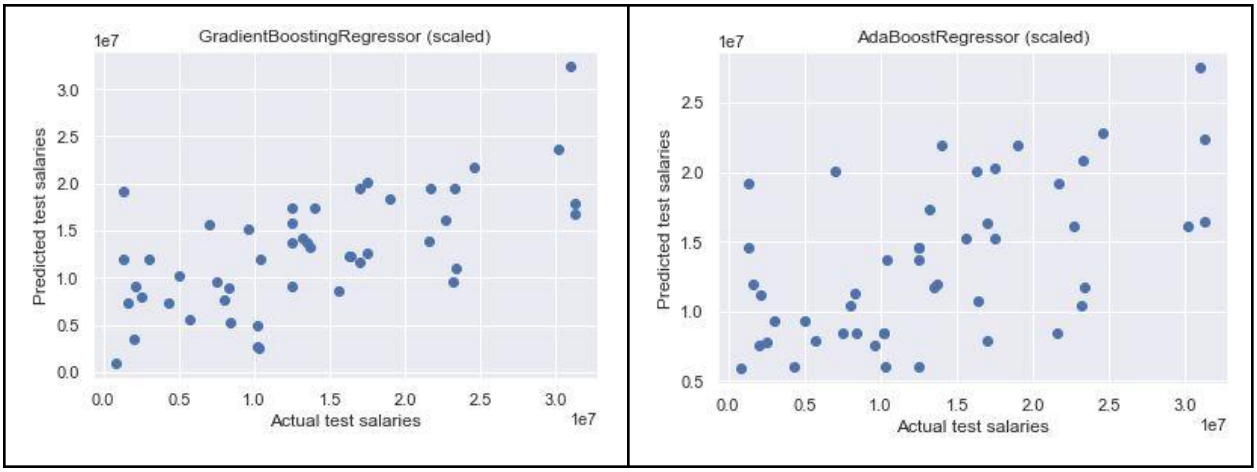
The next three models I tried were sklearn ensemble models, they were the RandomForestRegressor, GradientBoostingRegressor, and AdaBoostRegressor models.

models-*metrics*	*mean R^2 (GridSearchCV)*	*R^2 (train)*	*R^2 (test)*	*Adjusted R^2 (train)*	*Adjusted R^2 (test)*	*MAE (train)*	*MAE (test)*	*RMSE (train)*	*RMSE (test)*
Dummy (mean) Regressor	N/A	0	-0.001144622163	-0.1595744681	-0.4704311638	7067479.768	6951335.557	8290153.791	8507852.202
Linear Regression (unscaled)	N/A	0.6214726672	0.3892925341	0.5610693694	0.1030234095	3993532.992	4959487.834	5100481.887	6644895.294
Linear Regression (scaled)	N/A	0.6214726672	0.3892925341	0.5610693694	0.1030234095	3993532.992	4959487.834	5100481.887	6644895.294
Ridge Regression (scaled)	0.4327860265	0.5856605149	0.4230847569	0.3422731583	0.1526557367	4209099.691	4715319.277	5336306.281	6458438.588
Lasso Regression (scaled)	0.3745317665	0.6214726262	0.3893765573	0.2747230059	0.1031468186	3993490.105	4958987.241	5100482.162	6644438.164
ElasticNet Regression (scaled)	0.4327837711	0.5784262474	0.4226972406	0.342270543	0.1520865722	4255236.742	4710168.547	5382690.005	6460607.303
DecisionTreeRegressor (scaled)	0.3628601174	0.4798729114	0.1060998076	0.2611888595	-0.3129159076	4591332.135	5796544.278	5978845.4	8039259.411
RandomForestRegressor (scaled)	0.4407261668	0.9055482711	0.461732025	0.3514803424	0.2094189117	2008312.709	4598098.391	2547812.965	6238365.243
GradientBoostingRegressor (scaled)	0.4395221139	0.8855335992	0.4143208777	0.3500841534	0.1397837892	1054246.233	5030078.541	2804798.077	6507308.537
AdaBoostRegressor (scaled)	0.4576899585	0.7856629242	0.3119801129	0.3711511221	-0.01052920916	3257417.705	5425524.857	3838055.389	7052969.782

The random forest model required three hyperparameters: n_estimators, criterion, and bootstrap. Again I used GridSearchCV, and it determined the best combination values were n_estimators = 20, criterion = mae, and bootstrap = True. This model had the best metrics thus far, although the difference between its test prediction scores and those of the ridge model were not substantial. The gradient boosting model required hyperparameters n_estimators, loss, learning_rate, and max_depth which GridSearchCV determined to be 260, lad, 0.5, and 2 respectively as the best combination of values. This model did not perform better than the random forest model but did have a few metrics that performed better than the ridge model. Then I tried the ada boost model whose hyperparameters, using GridSearchCV, were determined to be n_estimators = 20, loss = linear, and learning_rate = 0.5. This model had a mean R^2 score of about 0.458, which was the best so far. The training data metrics were very good, but not so much for the testing data. In fact, it had some of the worst metrics regarding the testing data. This is a classic case of the model overfitting to the training data.

These were all the models that I tried and the three best performing ones were the ridge regression model, the elastic net model, and the random forest model. However, what I noticed about these models and all the other models was that they were very inaccurate with salaries below $3 million and salaries over $30 million. The scatter plots below show how well the predictions from each model match with the actual salaries. The x-axes represent the salaries in the test split while the y-axes represent the predicted salaries of the test split.


Linear Regression (unscaled)
Predicted test salaries
Actual test salaries
1e7
Ridge Regression (scaled)
Predicted test salaries
Actual test salaries
1e7
Lasso Regression (scaled)
Predicted test salaries
Actual test salaries
1e7
ElasticNet Regression (scaled)
Predicted test salaries
Actual test salaries
1e7
DecisionTreeRegressor (scaled)
Predicted test salaries
Actual test salaries
1e7
RandomForestRegressor (scaled)
Predicted test salaries
Actual test salaries
1e7



Most of them have a very obvious positive correlation but, as I stated earlier, all the models have a lot of trouble predicting salaries that are on either end of the spectrum of salaries. As a result, I removed the salaries that fall below $3 million and those that go above $30 million and used the three best performing model types on the remaining data.

Out of the 158 salaries that were in the original data, 29 were taken out because they didn't fall within the $3 to $30 million bounds. In theory, the models were now supposed to work much better but there are also other reasons for taking this data out that have to do with the way NBA teams operate. Generally speaking, salaries that fall under the $3 million threshold are for two types of players: players acquired late in a draft with very team-friendly (low-cost) contracts or players acquired after the vast majority of the roster has been constructed. Both of these types of players are typically low-impact, end-of-the-bench types that are relatively cheap. As for players that earn greater than $30 million, these types of contracts are given because teams have no choice regardless of the player's actual worth. If they are amazing all-stars, then it becomes a no-brainer. But if these players are not all-stars but believe they are, or even worse, if other teams believe they are, teams have two options. They can let the player walk in free agency and lose that player without getting anything in exchange (if they were signed already and their contract expired), or they could sign the player to a massive salary and hope they can play up to the value they signed for.

models-*metrics*	*mean R^2 (GridSearchCV)*	*R^2 (train)*	*R^2 (test)*	*Adjusted R^2 (train)*	*Adjusted R^2 (test)*	*MAE (train)*	*MAE (test)*	*RMSE (train)*	*RMSE (test)*
ElasticNet Regression (scaled)	0.3255089928	0.5455805423	0.4821033177	0.2178774491	0.2393392479	3644945.945	3290803.783	4477399.322	4227838.632
Ridge Regression (scaled)	0.3255239243	0.5530187069	0.4829340165	0.2178947633	0.2405593368	3609410.286	3275991.795	4440603.976	4224446.576
RandomForestRegressor (scaled)	0.2183300533	0.9154689561	0.4711292819	0.09359548732	0.2232211329	1474121.486	3419278.908	1931103.21	4272396.981

The above are the metrics for the three best models mentioned above, the elastic net model, ridge model, and the random forests model. I used them in order of worst performing to best performing and since the data was changing due to removing some players and rescaling, I used GridSearchCV on each again to find the new hyperparameter values. For the elastic net model, the hyperparameters were alpha = 1.5 and l1_ratio = 0.9. This elastic net model had a worse mean R^2 than the first iteration of the elastic net model, but the testing data metrics were much better. The MAE and RMSE were improved by over $1.4 and $2.2 million,

respectively. Then the ridge regression was tested, with hyperparameter alpha = 10.5. Just as with the elastic models, the new ridge model had a worse mean R^2 score than the first ridge models but had much better testing data metrics. The improvement in MAE and RMSE were almost the exact same with difference in scores being around $1.4 and $2.2 million also, respectively. Additionally, this new ridge model performed marginally better than the new elastic net model along all the metrics. Finally, I tried the random forest model with hyperparameter values of n_estimators = 50, criterion = mse, and bootstrap = True. This new random forest model had a mean R^2 of about 0.218, which is not only very bad but was the worst mean R^2 score of every model that was attempted on this project. The difference in MAE and RMSE between this new random forest model and the old were over $1.1 and $1.9 million, respectively. Although it did perform better than the old random forest model and even the new elastic net model, it could not outperform the new ridge model.

Now that this new ridge regression model has been established as the best performing model, we can take a look at how NBA teams can use it to inform their offseason decisions regarding free agent signings. The scatter plot below shows the testing split salaries predicted by the ridge model vs. the actual salaries in the testing split. As can be seen, the model tends to underpredict the salary of the player the vast majority of the time. This makes a lot of sense given that it is generally acceptable, and many times required, that teams overpay players for their services. That being said, this model can be used for 3 different purposes. First is the obvious, a team can use it to figure out how much they are overpaying or underpaying for a player and determine if the roster's overall production is enough to justify the bill. Most of the time the result will be that the team is overpaying, but an estimate of how much would be very helpful when roster decisions have to be made in the offseason. Second builds on the first, and that is to use it to gain insight into how much other teams are overpaying for their players and roster as a whole. Perhaps the opposing team being analyzed made a signing that isn't working out for them and consequently may want to get rid of via trade. The team can see if their overpaid players can be traded for another team's overpaid players to either save money or improve the team's production or both. Lastly, a team can use this model to gauge the trajectory of a player and determine just how much they'll have to overpay when they become a free agent. This will help them figure out how to best use their cap space now or in the future to make room for that free agent player.